VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

DIPLOMOVÁ PRÁCE

Implementace metody MapReduce

Implementation of MapReduce

2014 Bc. Martin Ševčík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Martin Ševčík**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Implementace metody MapReduce
Implementation of MapReduce

Zásady pro vypracování:

Cílem práce je prostudovat metodu MapReduce a její využíti pro distribuované výpočty a aplikovat ji na zvolený problém. Práce musí obsahovat:

1. Popis metody MapReduce.
2. Popis aplikací metody MapReduce.
3. Popis zvoleného problému.
4. Implementace metody MapReduce na zvoleném problému
5. Otestování efektivity řešení pro různé dimenze dat.

Seznam doporučené odborné literatury:

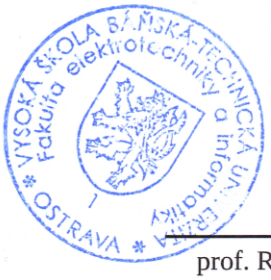
[1] Jeffrey Dean and Sanjay Ghemawat; MapReduce: Simplified Data Processing on Large Clusters, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.
[2] http://research.google.com/archive/mapreduce.html
[3] Tom White; Hadoop: The Definitive Guide, O'Reilly Media; Third Edition edition, 2012, ISBN-13: 978 -1449311520

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Platoš, Ph.D.**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

PROHLÁŠENÍ AUTORA DIPLOMOVÉ PRÁCE

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Ostrava 31. 7. 2014

Bc. Martin Ševčík

Poděkování

Na tomto místě bych rád poděkoval panu doc. Ing. Janu Platošovi, Ph.D., vedoucímu mé diplomové práce, za jeho ochotu, konzultace a odbornou pomoc.

ABSTRAKT

Diplomová práce je zaměřena na implementaci metody MapReduce. Cílem práce je naimplementovat model MapReduce a porovnat efektivitu této implementace s implementací na aplikačním rámci Hadoop. Nejprve je teoreticky popsán princip fungování programovacího modelu MapReduce, dále seznámení s open-source projektem Hadoop a návod na jeho spuštění v clusteru. V další části je popsán způsob implementace na dva různé problémy a otestována jejich efektivita pro různé dimenze dat.

Klíčová slova: MapReduce, Apache Hadoop, HDFS

ABSTRACT

Master's thesis is focused on implementation of MapReduce. The aim is to implement MapReduce model and compare effectiveness of this implementation with the implementation of the Hadoop framework. At first is theoretically described principle of functioning MapReduce programming model, also familiar with the open-source Hadoop project and instructions on how to run a Hadoop cluster. The next section describes how to deploy MapReduce on two different problems and tested their effectiveness for various dimensions of the data.

Keywords: MapReduce, Apache Hadoop, HDFS

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

Název	Anglický název	Popis
API	Application Programming Interface	Rozhraní pro programování aplikací
Atd.		A tak dále
CD	Compact Disc	Kompaktní disk
CUDA	Compute Unified Device Architecture	Hardwarová a softwarová architektura společnosti nVidia
ČR		Česká Republika
EMC forum		Konference zaměřená na Big Data
GB	Gigabyte	Jednotka množství dat v informatice, 10^9 Bajtů
GHz	GigaHertz	Jednotka frekvence, 10^9 Hertzů
GPU	Graphic processing unit	Grafický procesor
HDFS	Hadoop Distributed File System	Distribuovaný souborový systém
HPC	High-performance computing	Agregace výpočetního výkonu, pro dosažení vyššího výpočetního výkonu
HQL	Hive Query Language	Obdoba SQL s řadou funkčních omezení
IP		Jednoznačná identifikace zařízení v počítačové síti
ISBN	International Standard Book Number	Mezinárodní standardní číslo knihy
IT	Information technology	Informační technologie
JSON	JavaScript Object Nota-tion	JavaScriptový objektový zápis
JVM	Java Virtual Machine	Sada počítačových programů a datových struktur, která využívá modul virtuálního stroje ke spuštění dalších počítačových programů a skriptů vytvořených v jazyce Java.
MB	Megabyte	Jednotka množství dat v informatice, 10^6 Bajtů
Např.		Například

NoSQL	Not Only Structured Query Language	Databázový koncept
Obr.		Obrázek
OS		Operační systém
RAM	Random-access memory	Paměť s přímým přístupem
RPC	Remote procedure call	Vzdálené volání procedur
RSA		Šifra s veřejným klíčem
SQL	Structured Query Language	Standardizovaný dotazovací jazyk používaný pro práci s daty v relačních databázích
SSH	Secure Shell	Zabezpečený komunikační protokol
SVN	Apache Subversion	Systém pro správu a verzování zdrojových kódů
Tab.		Tabulka
Tzv.		Takzvaný
URI	Uniform Resource Identifier	Jednotný identifikátor zdroje
XML	Extensible Markup Language	Rozšiřitelný značkovací jazyk
YARN	Yet Another Resource Negotiator	Rámec pro plánování úkolů a řízení zdrojů clusteru.
č.		Číslo
http	Hypertext Transfer Protocol	Internetový protokol
char		Datový typ
id		Identifikace
int		Datový typ
log		Název pro záznam
www	World Wide Web	Soustava propojených hypertextových dokumentů

OBSAH

ÚVOD

Různá zařízení připojená k internetu generují značné množství dat, která je ne vždy jednoduché zpracovávat, analyzovat a využívat k dalším činnostem. Takzvaná big data jsou v současné době veliký pojem, se kterým pracuje stále více společností. Není žádná pevně stanovená hranice mezi daty a big daty. Technologickým vývojem se tato hranice posouvá a není proto vhodné hledat přesnou velikost. To co bylo považováno v roce 2008, kdy tento termín vznikl, za velká data, jsou dnes data malá. Velké společnosti na ukládání a zpracování big data zakládají budoucnost svého podnikání. Přínos big data je zjednodušeně charakterizován tak, že velké objemy dat, získané z různých zdrojů jsou zpracovány analytickými mechanismy a tím se objevují nejrůznější vztahy a závislosti, které jinak zjistit nejdou. Není ovšem výnosné tyto data ukládat a zpracovávat v běžných nástrojích a úložištích. Pro zlevnění a zefektivnění práce s takovými daty existuje několik řešení. Nejznámější z nich je Hadoop MapReduce.

Cílem diplomové práce je naimplementovat metodu MapReduce a porovnat efektivitu této implementace s implementací na aplikačním rámci Hadoop. V první kapitole je rozebrán pojem big data. Dále je teoreticky popsán princip fungování programovacího modelu MapReduce, jeho využití pro distribuované výpočty a některé jeho distribuce. Třetí kapitola obsahuje seznámení s open-source projektem Hadoop, který slouží pro zpracování a analýzu velkých objemů dat. Jsou zde popsány jeho jednotlivé části a princip fungování. Další část poskytuje ucelený návod pro přípravu, instalaci a spuštění Hadoop jak na jednom počítači, tak i v clusteru. V další kapitole je rozebrán způsob vlastní implementace metody MapReduce a implementace Hadoop na dvou různých úkolech. V poslední kapitole je na různých dimenzích dat otestována efektivita řešení těchto implementací.

.

1 BIG DATA

Fráze „Big data“ se poprvé objevila v oblasti high performance computingu (HPC). Jedná se o taková data, která jsou již mimo možnosti běžného zpracování nebo zachycení. Tento pojem se objevuje u HPC dodavatelů v souvislosti s vizualizačními platformami, cloudovými řešeními a úložišti. Definici toho, co vlastně již jsou nebo nejsou Big data, je hodně. Nejlépe tuto definici vystihuje poradenská firma Gartner: *Big data je termín aplikovaný na soubory dat, jejichž velikost je mimo schopnosti zachycovat, spravovat a zpracovávat data běžně používanými softwarovými nástroji v rozumném čase*. Pojem „velikost“ dat je chápán jak z hlediska objemu dat, tak i z hlediska rychlosti jejich tvorby a různorodosti jejich typů. Nástup webu, mobilních zařízení a dalších technologií způsobil zásadní změnu charakteru dat a metod jejich využití. Již nejsou centralizovaná, vysoce strukturovaná a snadno zvládnutelná, ale více než dříve jsou volně strukturovaná, vysoce distribuovaná a mají vzrůstající objem.

Problém Big data může být rozdělen do tří základních rovin:

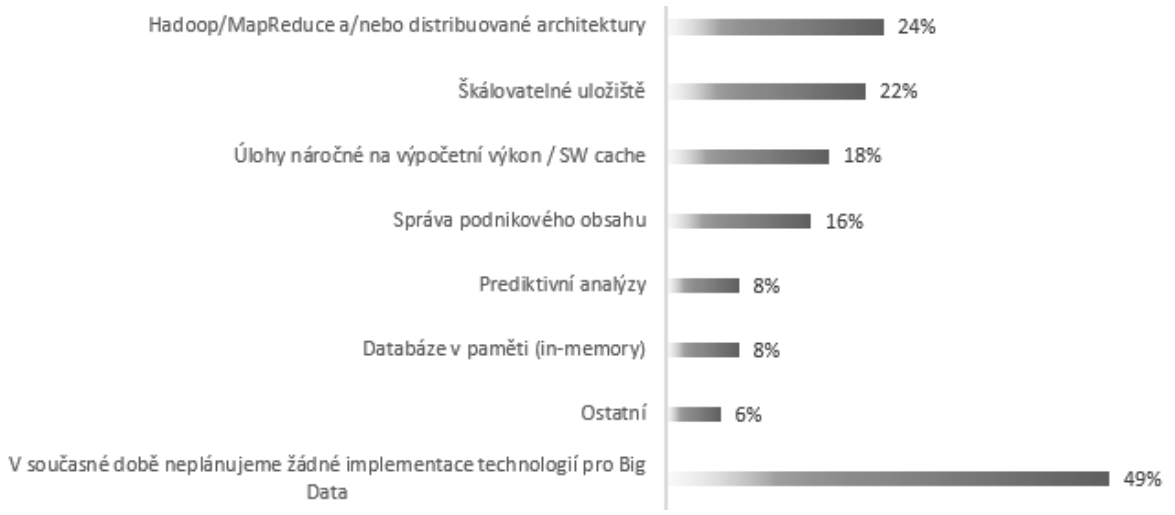
- Objem (volume) – Množství dat vznikajících v rámci provozu firem roste exponenciálně každý rok.
- Typ (variety) – Různorodost typů dat vzrůstá, například nestrukturované textové soubory, semi-strukturovaná data (XML), data o geografické poloze, data z logů.
- Rychlost (velocity) – Rychlost s jakou data vznikají a potřeba jejich analýzy v reálném čase vzrůstá díky pokračující digitalizaci většiny transakcí, mobilním zařízením a vzrůstajícímu počtu internetových uživatelů.

Podle toho, o jakou z těchto tří oblastí jde primárně, se pak liší jednotlivé přístupy. Nejde jen o problém softwarový, ale zasahuje také do hardwaru nebo optimalizace vzorkování dat. [1,2]

1.1 Big data v ČR

Přestože se podnikům hromadí objemy strukturovaných i nestrukturovaných dat v různých podobách a roste potřeba z nich těžit významné informace pro rozhodovací procesy, implementace velkých dat jsou většinou v experimentální nebo testovací fázi. Pro dosažení maximální efektivity vyžaduje každý případ odlišnou kombinaci softwaru, hardwaru a služeb, a navíc individuální úpravy s ohledem na oblast působnosti a typ obchodních aktivit. Velký zájem je hlavně v odvětvích bankovnictví, pojišťovnictví, telekomunikacích a maloobchodu, které si uvědomují potenciální obchodní přínosy ve složitém ekonomickém prostředí. Mnoho společností již zahájilo virtualizaci a využívá cloudové technologie. Některé dokonce vytvořily plně softwarově definovanou infrastrukturu. Tento přístup spolu s analytickými technologiemi big data přinesou silný růst konkurenceschopnosti českých podniků. [3,4]

Následující obrázek č. 1 obsahuje výsledky průzkumu, který probíhal v České republice na konferenci EMC Forum 2013 v Praze. Cílem bylo zjistit nejčastěji využívané nástroje pro práci s big daty. V rámci tohoto průzkumu bylo dotazováno 237 pracovníků s rozhodovací pravomocí v oblasti IT. [3]



Obr. 1. Nejčastěji využívané nástroje pro práci s big daty [3]

K nejoblíbenějším nástrojům v českých firmách, které je používají nebo plánují nasadit, patří architektura Hadoop/MapReduce a distribuované architektury. Organizace, které tyto nástroje dosud nenasadily nebo to vůbec neplánují, uvádějí několik důvodů. Nejčastější je, že tyto nástroje nejsou v daném oboru relevantní, nebo že neexistuje žádný obchodní případ či prokázaná návratnost investice, popřípadě jsou příliš nákladné. [3]

Big data mají vlastnosti, které je odlišují od obvyklých firemních dat. Tradiční datové sklady a nástroje pro správu dat nejsou připraveny na zpracování a analýzy velkých objemů dat ve velmi krátkém čase nebo nákladově efektivním způsobem. Proto je třeba hledat nové způsoby zpracování a analýzy velkých objemů dat. Jedním z takových způsobů je MapReduce. [2]

2 MAPREDUCE

MapReduce je programovací model pro vyjádření distribuovaných výpočtů na velké množství dat a slouží i jako rámec pro spouštění rozsáhlých zpracování dat na clusterech distribuovaných serverů. Uživatel definuje funkci Map, která z dvojic (klíč, hodnota) vygeneruje pracovní dvojice (klíč, přechodná hodnota) a funkci Reduce, která z klíče a seznamu přechodných hodnot vytvoří výslednou hodnotu tohoto klíče. Programy napsané tímto způsobem se automaticky paralelizují a jsou prováděny ve velkých clusterech. Run-time systém se stará o rozdělení vstupních dat, plánování provádění programu mezi počítači, zpracování chyb a zprávu požadované komunikace. To umožňuje programátorům bez jakýchkoliv zkušeností s paralelními a distribuovanými systémy snadno využívat zdroje velkých distribuovaných systémů. Typický MapReduce výpočet zpracovává mnoho terabajtů dat na tisíci počítačích. [13, 18]

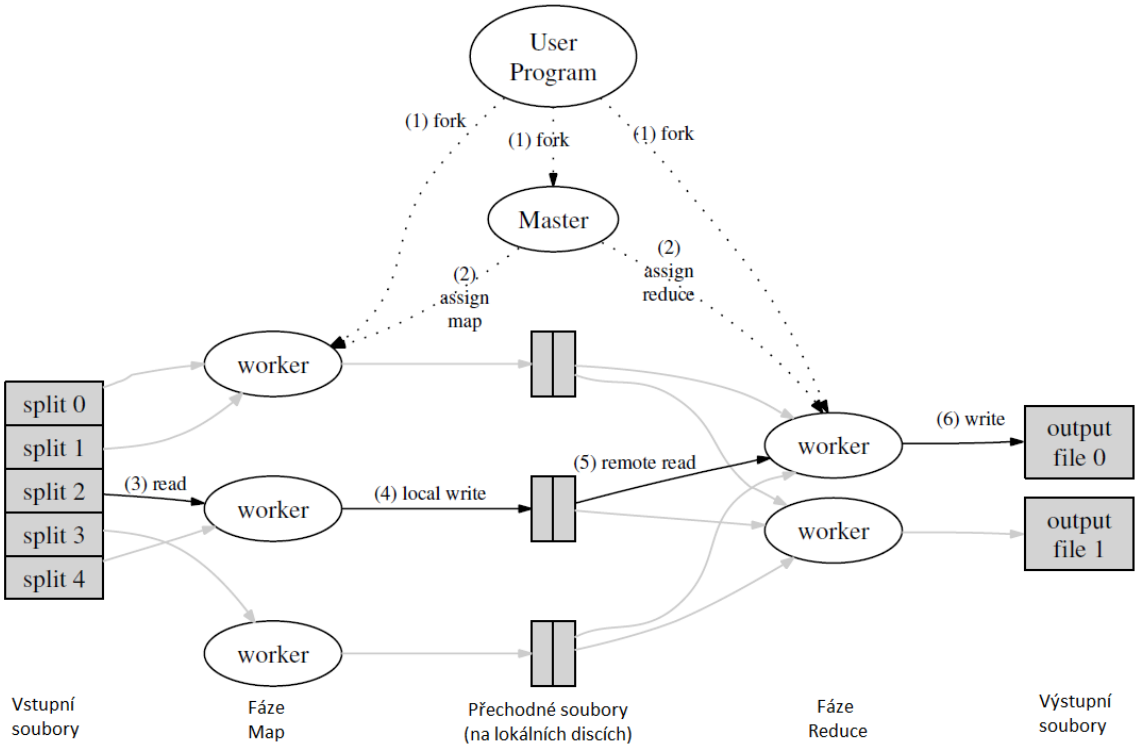
2.1 Princip MapReduce

Výpočet bere sadu vstupních dvojic (klíč, hodnota) a vytváří sadu výstupních dvojic (klíč, hodnota).

Uživatel knihovny MapReduce vyjadřuje výpočet jako dvě funkce Map a Reduce.

- Funkce Map, bere vstupní pár a vytváří soubor přechodných párů klíč / hodnota. MapReduce knihovna sdružuje všechny přechodné hodnoty spojené se stejným klíče a předává je funkci Reduce.
- Funkce Reduce, přijímá přechodný klíč a sadu hodnot pro tento klíč. Tyto hodnoty spojuje dohromady pro vytvoření případně menší sady hodnot. Přechodné hodnoty jsou dodávány do uživatelské funkce Reduce pomocí iterátoru. To umožňuje zpracovávat seznamy hodnot, které jsou příliš velké na to, aby se vešli do paměti.

Volání funkce Map je distribuované na několika samostatně běžících počítačích. Distribuce je realizována pomocí rozdělení vstupních dat na M částí. Tyto rozdělené části můžou být zpracovány paralelně na různých počítačích. Volání funkce Reduce je distribuované pomocí rozdělení prostoru klíčů na R částí, nejčastěji pomocí hashovací funkce (*hash*(*key*) **mod** *R*). Počet oddílu R a dělící funkce jsou definované uživatelem. [13]



Obr. 2. Přehled MapReduce činnosti [13]

Na obrázku č. 2 je zobrazen celkový průběh operace MapReduce v implementaci firmy Google. Při zavolání funkce MapReduce uživatelským programem dojde k následující posloupnosti akcí:

1. MapReduce knihovna nejprve v uživatelském programu rozdělí vstupní soubory na M částí. Velikost souboru je kontrolovaná uživatelem pomocí volitelného parametru. Framework poté spustí mnoho kopií programu na více oddělených počítačích.
2. Jedna ze spuštěných kopií programu je odlišná od ostatních. Její název je Master a stará se o přidělování úloh pro ostatní uzly. Ostatní kopie jsou pracovní uzly, kterým master přiřazuje úkoly. Master se také stará o koordinaci celé práce, ve které přiřazuje nevyužitým uzlům *M* map úkolů a *R* reduce úkolu.
3. Pracovní uzel, který je přidělen k provádění mapovací úlohy nejprve přečte obsah příslušné části vstupu. Analyzuje dvojici <klíč, hodnota> ze vstupních dat a předá každý pár uživatelem definované Map funkci. Výsledkem Map funkce jsou dočasné páry <klíč, přechodná hodnota>, které jsou uloženy do vyrovnávací paměti.
4. Dvojice uložené ve vyrovnávací paměti jsou v pravidelných intervalech na jednotlivých pracovních uzlech zapisovány na lokální disky. Umístění těchto dvojic je posláno uzlu Master, který zodpovídá za přeposlaní umístění dvojic na pracovní uzly vykonávající Reduce úkoly.
5. Jakmile pracovní uzel provádějící Reduce úkoly dostane zprávu o umístění dat, použije vzdálené volání procedur pro načtení uložených dat. Po přečtení všech přechodných dat je seřadí podle přechodných klíčů tak, aby všechny data se stejným klíčem byly spolu seskupené. Třídění je nutné, protože je obvykle na vstupu jednoho Reduce uzlu mnoho dvojic

s rozdílnými klíči. Pokud je množství přechodných dat příliš velké a nevleze se do paměti, pak se použije externí třídění.

6. Uzel Reduce přechází přes seřazená přechodná data a pro každý jedinečný přechodný klíč zavolá funkci Reduce, které předá klíč a soubor odpovídajících hodnot. Výsledek funkce Reduce se připojí do konečného výstupního souboru pro tuto část přechodných dat.
7. Jakmile jsou všechny úlohy dokončeny, tak se pokračuje ve vykonávání programu, odkud byl MapReduce volaný.

Po úspěšném dokončení je výstup MapReduce úlohy dostupný v R výstupních souborech. Uživatelé obvykle nepotřebují těchto R výstupních souborů spojit do jednoho souboru, ale předávají je jako vstup dalšího volání MapReduce, nebo se použijí u jiných distribuovaných aplikací, které jsou schopné řešit vstup rozdělený do několika souborů. [13]

2.1.1 Obsluha chyb

Vzhledem k tomu, že je knihovna MapReduce navržena pro zpracování velkého množství dat na mnoho počítačích, musí proto vhodně zpracovat selhání jednotlivých uzlů.

- ***Selhání pracovního uzlu***

Uzel Master pravidelně kontroluje pracovní uzly. Pokud do určitého času nedostane odpověď, označí daný pracovní uzel jako chybný. Úloha prováděná chybný uzlem je vrácena do původního nezpracovaného stavu a přidělena jinému uzlu. V případě chyby u již dokončené Map úlohy se musí provést znovu, protože přechodné výsledky jsou uloženy na lokálních discích jednotlivých uzlů. U dokončených Reduce úloh tohle neplatí, protože se jejich výsledky zpravidla zapisují na globální distribuovaný souborový systém. V případě chyby dostanou všechny pracovní uzly zprávu o změně. MapReduce je díky tomuto postupu odolný vůči rozsáhlým výpadkům pracovních uzlů.

- ***Selhání uzlu Master***

Uzel Master pravidelně vytváří kontrolní body. Pokud selže, tak se zahájí nová kopie z posledního kontrolního bodu. V případě jako je na obrázku 2, kde je pouze jeden uzel Master se výpočet přeruší. Klienti si můžou zjistit stav a poté opakovat operaci MapReduce. [13]

2.1.2 Příklad

Jeden z příkladů, může být výpočet množství výskytu každého slova ve velké kolekci dokumentů. Výsledný kód by se podobal následujícímu pseudokódu. Tento příklad je rozebrán i v kapitole č. 5.

```
map(String key, String value):
      // key: název dokument
      // value: obsah dokumentu
      for each word w in value:
            EmitIntermediate(w, key);
reduce(String key, Iterator values):
      // key: slovo
      // values: seznam výskytů
      int result = 0;
      for each v in values:
            result += ParseInt(v);
      Emit(AsString(result));
```

Funkce Map vydává (emituje) každé slovo, spolu se souvisejícím počtem výskytů.

Funkce Reduce sečte dohromady všechny počty výskytů vydané pro určité slovo. [13]

Vhodné typy problémů pro MapReduce jsou:

- vytváření indexu
- strojové učení na velkých datech (automatický překlad)
- extrahování dat
- statistiky a analýza dat
- clusterizace
- archivace a zálohování dat
- manipulace s velkými grafy.

2.2 Dostupné implementace

MapReduce se původně odkazoval na patentové technologie Google. Postupem času vznikly open source implementace, které jsou napsané v mnoha programovacích jazycích. Jejich využití, architektury nebo hardwarové platformy se liší. Výhody těchto implementací jsou především jednoduchost, vysoká úroveň paralelismu, velká škálovatelnost a nenáročnost na hardware. Existuje celá řada těchto implementací. Například:

- Apache Hadoop

  Jedná se o rámec, který umožňuje distribuované zpracování velkého objemu dat na clusterech počítačů pomocí jednoduchých programovacích modelů. Patří mezi nejpopulárnější volně dostupné implementace a je taky rozebrána v další kapitola. [5]

- Disco

  Disco je odlehčený, open source rámec pro distribuované výpočty na základě MapReduce paradigmatu. Tento rámec vznikl v Nokia Research Center v roce 2008, pro řešení reálných

problémů při manipulaci velkého množství dat. Díky dynamickému objektově orientovanému skriptovacímu jazyku Python je výkonný a snadno použitelný. Disco distribuuje a replikuje data a efektivně rozvrhuje práci. Obsahuje nástroje pro indexaci velkého množství dat v řádu miliard. Používá se pro různě účely jako log analýzy, pravděpodobnostní modelování nebo fulltextové indexování. [6]

- Meguro

  Tento MapReduce rámec je postaven na základech Tokyo Cabinet (knihovna sloužící pro řízení databáze) a Tracemonkey (zajišťuje logiku MapReduce). Jako inspirace slouží javascriptová MapReduce knihovna obsažená v CouchDB. Základní myšlenka přišla z potřeby, snadno pracovat s přiměřeně velkými datovými soubory, které se skládaly především z JSON uvnitř databáze Tokyo Cabinet. Meguro naimplementuje většinu složité práce a uživatel pouze naimplementuje knihovny a volání Map a Reduce. [7]

- Octopy

  V případě Octopy se jedná o rychlou a snadnou implementaci v Pythonu. Datovou sémantikou je inspirován od Google MapReduce a rozsahem projektu je více podobný na utilitu pro distribuované programování StarFish. Octopy si neklade za cíl splnit všechny možnosti distribuovaných výpočtů, ale umožňuje jednoduchým způsobem změnit velkou část úkolů na paralelizované. [8]

- Mars

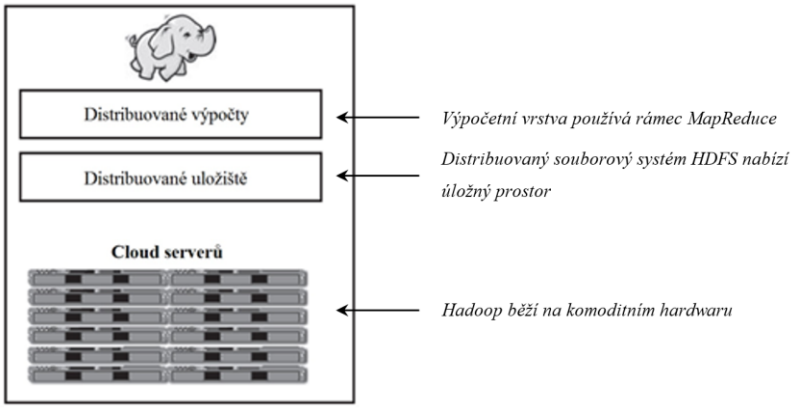
  Na rozdíl od ostatních uvedených implementací je tento MapReduce rámec pro grafické procesory (GPU). Cílem tohoto rámce je poskytnout obecný rámec pro vývojáře, který zajišťuje implementaci dat a výpočetně náročné úkoly efektivně na GPU. Mars skrývá složitost programování GPU za jednoduché a známé MapReduce rozhraní. Proto mohou vývojáři napsat svůj kód na GPU bez znalosti grafického API rozhraní nebo GPU architektury. Mars je implementován pomocí nVIDIA CUDA. [9]

- A mnoho dalších…

Přehled dalších distribucí a jejich vlastností k červenci 2014 je na obrázku č. 30, v příloze č. 1.

3 APACHE HADOOP

Hadoop je open source framework pro zpracování, ukládání a analýzu velkého množství distribuovaných, nestrukturovaných dat. Původně byl vytvořen ve společnosti Yahoo!. Jako inspirace byla použita MapReduce, uživatelsky definovaná funkce vyvinutá společností Google pro indexování webu. Hadoop je stavěn pro zvládání petabajtů a exabajtů dat distribuovaných přes více uzlů současně. MapReduce je výpočetní vrstva v rámci Hadoop. Úlohy MapReduce přistupují k datům, která jsou distribuována na webu nebo v datových centrech, rozdělují je do více replikovaných dílů a jejich zpracování pošlou na jednotlivé uzly. Dotazy a další zpracování pak probíhá v každém uzlu paralelně. Výsledky jsou agregovány a ukládány do úložné vrstvy, jako například Hadoop Distributed File System (HDFS). Odtud jsou data načtena do jednoho z několika analytických prostředí pro analýzu. [2]



Obr. 3. Prostředí Hadoop [14]

Projekt Hadoop zahrnuje následující moduly:

- Hadoop Common: Obsahuje společné nástroje, které podporují další Hadoop moduly.
- Hadoop Distributed File Systém (HDFS): Distribuovaný souborový systém, který poskytuje přístup s vysokou propustností k datům aplikace.
- Hadoop YARN (Yet Another Resource Negotiator): Rámec pro plánování úkolů a řízení zdrojů clusteru.
- Hadoop MapReduce: Systém založený na YARN, pro paralelní zpracování velkých datových souborů. [5]

Mimo tyto hlavní moduly existuje celá řada souvisejících projektů, které poskytují další nástroje pro prací s daty v Hadoop. Jsou to například:

Nástroje pro vývoj

- Hive – Jazyk vyvinutý společností Facebook umožňující psát dotazy v HQL, což je obdoba klasického SQL. HQL dotaz je pak Hive službou přeložen na MapReduce úlohy.
- Pig – Vyvinut společností Yahoo!, pro větší soustředění na analýzu dat a méně psaní Map-Reduce úloh. Skládá se ze dvou částí: první je PigLatin, tedy jazyk sám a druhým je běhové prostředí, ve kterém jsou programy v PigLatin jazyce překládány na sérii MapReduce úloh.
- Jaql – Deklarativní dotazovací jazyk pro JavaScript Object Notation (JSON) umožňující vybrat, spojovat, agregovat a filtrovat data uložená v HDFS.
- Mahout – Škálovatelná knihovna pro strojové učení. Jsou v ní implementovány algoritmy pro clustering, klasifikaci a filtrování optimalizované pro běh v prostředí Hadoop.

Ukládání dat a správa metadat

- Cassandra – Nejedná se o souborový systém, ale o NoSQL (klíč-hodnota) úložiště. Je alternativou k HDFS v aplikacích, které vyžadují rychlý přístup k datům z/do Hadoopu.
- HBase – Sloupcově orientovaný databázový systém, který běží nad HDFS, a který je vhodný pro řídké soubory dat. Na rozdíl od relačních databázových systémů, HBase nepodporuje strukturovaný dotazovací jazyk (jako např. SQL), ve skutečnosti HBase není ani relační úložiště dat. HBase aplikace jsou psány v Javě stejně jako typické MapReduce aplikace.
- HCatalog – Systém pro správu tabulek a úložišť v Hadoop, který umožňuje uživatelům s různými systémy pro zpracování dat (Pig, MapReduce a Hive) snadněji číst a zapisovat data.

Řízení

- Zookeeper – Koordinační služba pro distribuované aplikace. Vystavuje služby pro správu konfigurace, pojmenování, synchronizace a skupinové služby.
- Oozie – Systém pro správu Hadoop úloh (workflow scheduler).

Agregování a získávání dat

- Sqoop – Nástroj určený pro efektivní přenos hromadných dat mezi Hadoop a strukturovanými datovými sklady, jako jsou relační databáze.
- Chukwa – Systém sběru dat pro správu rozsáhlých distribuovaných systémů. Je postaven na HDFS a MapReduce frameworku. Obsahuje flexibilní a výkonné nástroje pro zobrazování, sledování a analyzování výsledků ze shromážděných dat.
- Flume – Služba pro distribuované přenosy, shromažďování a agregování velkého množství dat z datových logů. [26]

3.1 Hadoop MapReduce

Hadoop MapReduce je speciální implementace programovacího modelu MapReduce a výpočetním prvkem projektu Apache Hadoop. Poskytuje programátorům schopnost produkovat paralelní distribuované aplikace mnohem snadněji tím, že stačí napsat pouze funkce map() a reduce(), které řeší logiku specifického problému, zatímco framework MapReduce se automaticky stará o řízení distribuovaných serverů, paralelizaci úloh, řízení veškeré komunikace a datových přenosů mezi různými částmi systému a zajištění redundance v případě selhání. Je podobná tradičním distribuovaným výpočetním systémům a poskytuje démony JobTracker a TaskTracker. Démon je v informatice označení programu, který čeká na událost, kterou obslouží bez nutnosti uživatele. Tyto dva démoni zpracovávají řízení MapReduce úloh. Obecně platí, že je jeden JobTracker na cluster a více TaskTrackerů na uzly v clusteru. Přestože je rámec Hadoop implementován v jazyce Java, MapReduce aplikace můžou být implementovány i v jiných jazycích. [5, 14, 17, 26]

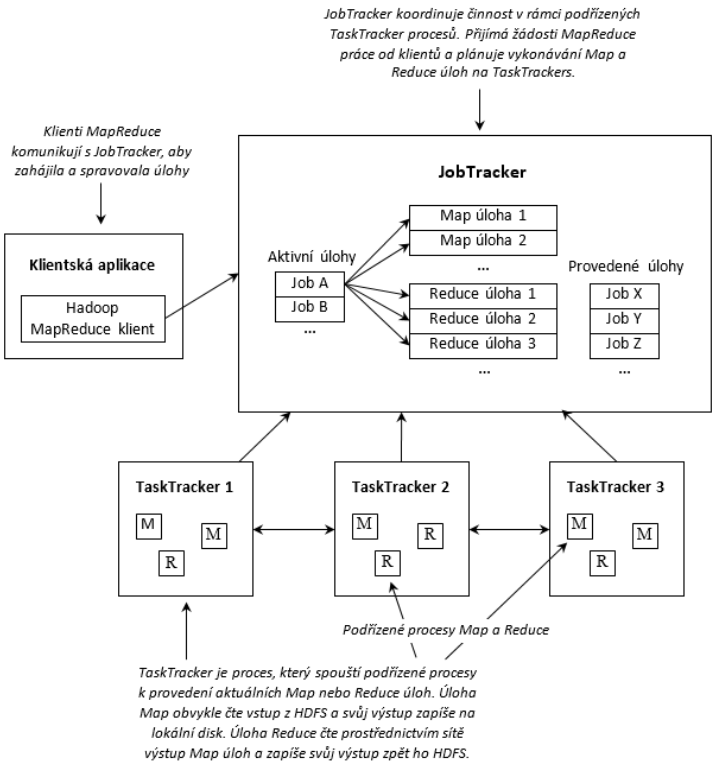
3.1.1 JobTracker

Jedná se o hlavního démona, který je zodpovědný za přijetí práce od klientů, plánování provozu na pracovních uzlech a poskytování administrativních funkcí (např. funkčnost pracovního uzlu nebo sledování vývoje úkolů na clusteru). K dispozici je jeden JobTracker na MapReduce cluster, který obvykle běží na spolehlivém hardwaru. V případě jeho selhání by totiž mělo za následek selhání všech spuštěných úloh. Klienti a TaskTrackers komunikují s tímto démonem pomocí vzdáleného volání procedur (RPC). [5,16,17]

3.1.2 TaskTracker

Druhým démonem je TaskTracker. Tento démon přijímá úkoly z JobTrackeru, vytvoří instanci uživatelského kódu, tyto úkoly lokálně vykonává a pravidelně podává zprávy JobTrackeru o pokroku zpracování. Na pracovním uzlu je vždy jeden TaskTracker, který je konfigurován s určitým počtem slotů pro map a reduce úkoly, které ukazují, kolik z každého typu úkolů je schopen provádět paralelně. V případě slotu se jedná o přidělení dostupných zdrojů na pracovním uzlu, ke kterému může být úkol přiřazen a proveden. Map a Reduce úkoly jsou konfigurovány odděleně, jelikož každý využívá zdroje odlišně. Obvykle TaskTracker umožňuje paralelně spouštět více Map úkolů než Reduce úkolů. Toto rozdělení je důležité pro plné využití hardwaru pracovního uzlu. [16]

Komunikace obou démonu je popsána na obrázku č. 4.



Obr. 4. Logická architektura MapReduce [14]

3.2 HDFS

HDFS (Hadoop Distributed File System) je distribuovaný souborový systém, který je určen pro vykonávání MapReduce úloh a slouží jako uložiště využívané aplikacemi Hadoop. Je optimalizován pro vysokou propustnost a nejlépe funguje při čtení a zápisu velkých souborů. Hlavní myšlenkou je rozdělení uživatelských dat do bloků (kousky souboru) a replikovat tyto bloky přes lokální uzly disků v clusteru. Rozdělení dat do bloků není nová myšlenka, ale HDFS bloky jsou výrazně vetší (64 MB ve výchozím nastavení) než velikost bloku v typickém souborovém systému. Distribuovaný systém přejímá master/slave architekturu, kde master spravuje souborový jmenný prostor a slave spravuje konkrétní datové bloky. [14,17,18]

HDFS služby jsou poskytovány třemi démony, viz tabulka č. 1.

Démon	Počet na cluster	Účel
Namenode	1	Ukládá metadata souborového systému a poskytuje globální obraz souborového sytému.
Sekundární namenode	1	Provádí vnitřní evidenci transakcí namenodu.
Datanode	Mnoho	Ukládá a načítá datové bloky (obsah souboru).

Tab. 1. HDFS démoni [16]

3.2.1 Namenode

Namenode je démon, který uchovává metadata (dodávané výhradně z RAM paměti pro rychlé vyhledávání a načítání) a udržuje úplný obraz souborového systému. Chce-li uživatel vytvořit nový soubor a zapisovat data na HDFS, pak klientská aplikace nejprve zkontaktuje Namenode. Ten po kontrole oprávnění aktualizuje souborový jmenný prostor a ujistí se, zda soubor již neexistuje. Poté na vhodném datanodu alokuje nový blok, na který bude aplikace posílat data.

HDFS namenode obstarává následující úkoly:

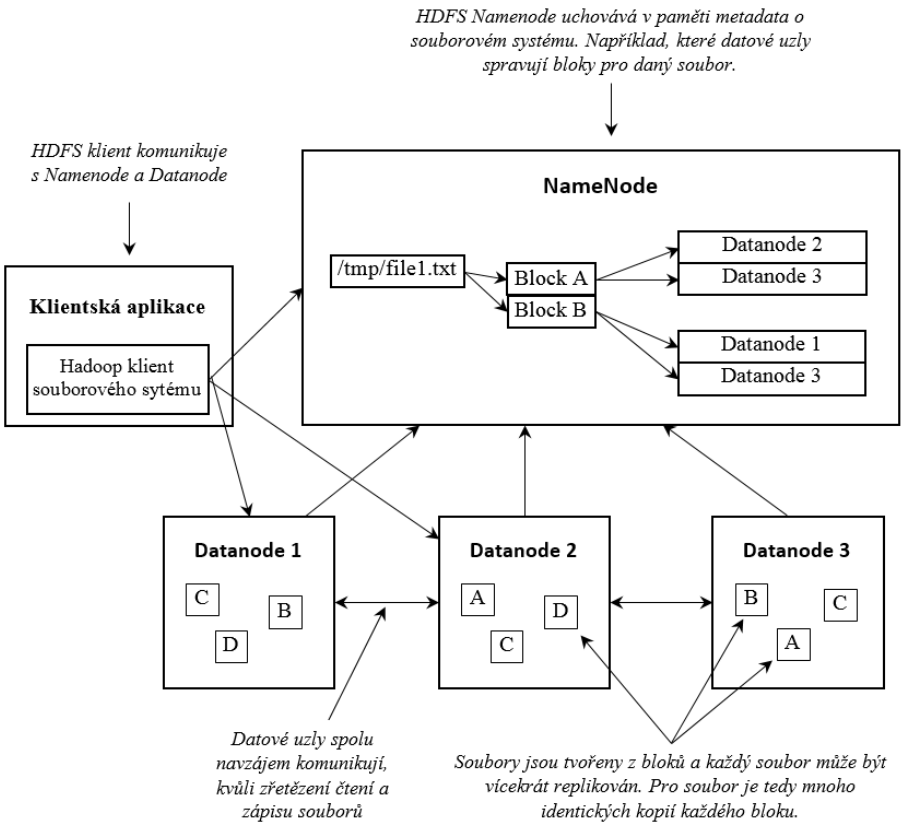
- Správa jmenného prostoru
  Namenode je zodpovědný za správu souborového jmenného prostoru, který obsahuje metadata, adresářovou strukturu, bloky k mapování, umístění bloků a přístupová oprávnění. Tato data jsou uložena do paměti s rychlým přístupem.
- Koordinace práce se soubory
  Namenode směřuje klienty aplikace na datanode, kde provádějí čtení a přiděluje bloky na vhodný datanode, kde se uloží. Všechny datové přenosy jsou přímo mezi klientem a datanode.
- Údržba celkového stavu souborového systému
  Namenode je v kontaktu s datanode prostřednictvím pravidelných zpráv k zajištění integrity systému. Má tedy přehled o všech datanode v clusteru. Pokud namenode zaznamená, že je méně kopií datového bloku uložených v datanode, než je požadovaný faktor replikace. Pak zařídí vytvoření nových replik. Je i zodpovědný za vyvážení souborového systému. V průběhu běžného provozu může datanode skončit s více bloky než ostatní. Vyvážení spočívá v přesunutí těchto bloků k datanode s méně bloky. To vede k lepšímu vyrovnávání zátěže a využití disku. [16,18]

Je taky možné spustit *sekundární namenode*, který se i přes svůj název nechová jako namenode. Jeho úkolem je pravidelně slučovat obraz jmenného prostoru spolu s protokolem změn pro případ, že by se tento protokol stával příliš velkým. Sekundární namenode zpravidla běží na samostatném fyzickém počítači.

3.2.2 Datanode

Jedná se o démona, který je zodpovědný za ukládání a načítání datových bloků v HDFS. Datanode má přímý lokální přístup k jednomu nebo více datovým diskům na serveru. V provozu jsou tyto disky obvykle vyhrazeny výhradně pro Hadoop. Při spuštění datanodu se pravidelně odesílá zpráva o blocích namenodu. Jedná se o seznam všech bloků, které v současné době má datanode na svých discích a umožňuje tak sledovat jakékoliv změny. [16]

Fungování HDFS architektury je na obrázku č. 5.

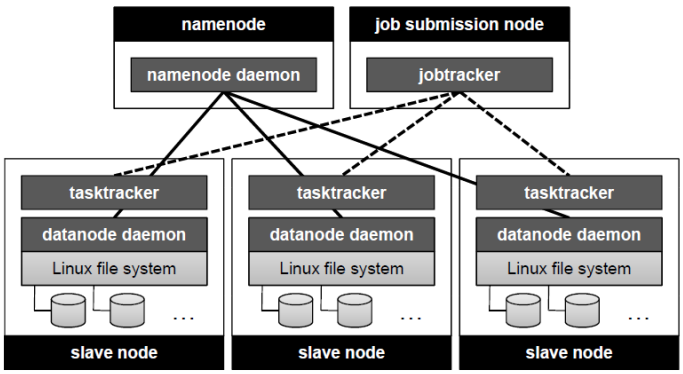


Obr. 5. HDFS architektura [14]

3.3 Kompletní architektura Hadoop

Kompletní architektura Hadoop clusteru (obr. číslo 6), se skládá ze tří části:

- HDFS master jako tzv. namenode
- Job submission node jako tzv. jobtracker
- Slave node (podřízený uzel), kde každý z těchto uzlů spustí tasktracker pro provádění map a reduce úkolů a démona datanode pro manipulaci s HDFS daty



Obr. 6. Architektura kompletního Hadoop clusteru [18]

4 SPUŠTĚNÍ HADOOP

Tato kapitola popisuje přípravu, instalaci a spuštění Hadoop na OS Ubuntu. Hadoop podporuje 3 provozní režimy, podle kterých je ovlivněno nastavení konfiguračních souborů. Nejprve je popsáno nastavení a spuštění Hadoop s jedním uzlem v tzv. pseudo-distribuovaném režimu. Dále pak změna v nastavení tak, aby Hadoop běžel v clusteru s více uzly, v plně-distribuovaném režimu.

- Local (Standalone) Mode
  Samostatný režim je výchozí režim pro Hadoop. Má prázdné konfigurační soubory a běží zcela na lokálním počítači. Nekomunikuje s ostatními uzly, nepoužívá HDFS a nespouští žádné démony. Tento mód nebere ohled na nastavení hardwaru a je vhodný pro ladění.
- Pseudo-Distributed Mode
  Hadoop je možné spustit na jednom uzlu v pseudo-distribuovaném režimu, kde každý proces běží odděleně na jednom počítači. Tento režim doplňuje samostatný režim o ladění kódu. To umožňuje prozkoumat využití paměti, HDFS vstupní/výstupní problémy a další interakce mezi procesy.
- Fully-Distributed Mode
  Jedná se o netriviální režim, kde se konfiguruje a spravuje od několika uzlů na clusteru po extrémně velké uzly s tisíce clustery. [12,15]

4.1 Verze Hadoop

Při psaní této práce existuje několik aktivních verzí, které jsou ke stažení na oficiálních stránkách Apache Hadoop. Rozdíly mezi jednotlivými verzemi jsou vypsány v tabulce č. 2.

- 1.2.x – Aktuální stabilní verze, vydání 1.2.
  Jedná se o pokračování verze 0.20. V této práci je používána verze 1.2.1, protože v době vypracování byla označována za nejstabilnější.
- 2.2.x – Aktuální stabilní 2.x verze.
  Toto vydání bylo označováno jako beta verze, ale postupem času (15. 10. 2013) se rovněž stalo stabilní verzí. Obsahuje několik nových funkcí oproti verzi 1.2.x.
  - Nový MapReduce runtime, nazvaný MapReduce 2, implementovaný na systému YARN. Rozděluje dvě hlavní funkce JobTrackeru: řízení zdrojů a správu životního cyklu úloh do samostatných komponent.
  - HDFS federace, která rozděluje jmenný prostor HDFS přes několik namenodes pro podporu clusterů s velkým počtem souborů.
  - HDFS s vysokou dostupností, který odstraňuje namenode jako jediný bod selhání podporou namenodes v pohotovostním režimu, které při selhání převezmou služby.
- 0.23.x – Obdoba 2.2.x, ale neobsahuje NameNode s vysokou dostupností. [5,19]

Vlastnost	1.2.x	0.23.x	2.2.x
Zabezpečené ověřování	Ano	Ano	Ano
Staré názvy konfigurace	Ano	Ne	Ne
Nové názvy konfigurace	Ne	Ano	Ano
Staré MapReduce API	Ano	Ano	Ano
Nové MapReduce API	Ano (chybí některé knihovny)	Ano	Ano
MapReduce 1 runtime	Ano	Ne	Ne
MapReduce 2 runtime (YARN)	Ne	Ano	Ano
HDFS federace	Ne	Ano	Ano
HDFS s vysokou dostupností	Ne	Ne	Ano

Tab. 2. Funkce podporované Hadoop verzemi [19]

4.2 Požadavky před instalací

Hadoop potřebuje funkční instalaci Javy. Doporučuje se Java 7 nebo vyšší verze Javy 6. Instalace Javy a následná kontrola nainstalované verze se provede následovně:

```
$ sudo apt-get install openjdk-7-jdk
$ java –version
```

4.2.1 Vytvoření Hadoop uživatele

Pro provoz Hadoop je doporučeno používat vyhrazený uživatelský účet. Slouží hlavně k oddělení instalace Hadoop od jiných softwarových aplikací a uživatelských účtů běžících na stejném počítači. Tento účet (v této práci je používán **hduser**) je určen pouze pro správu clusteru. Jakmile jsou spuštěni všichni démoni clusteru, můžou se MapReduce úlohy spustit z jiných účtů. [10,15]

Vytvoření skupiny hadoop: `$ sudo addgroup hadoop`

Přidání uživatele **hduser** do skupiny hadoop: `$ sudo adduser --ingroup hadoop hduser`

Přepnutí na nově přidaného uživatele: `$ su – hduser`

4.2.2 Konfigurace SSH

Hadoop vyžaduje SSH přístup ke správě svých uzlů na vzdálených počítačích. Jelikož je popisováno single-node nastavení, je třeba pro uživatele **hduser** nakonfigurovat SSH přístup k localhostu. Prvním krokem je zjistit, zda je SSH naistalován na uzlu. To se provede příkazem *which ssh*, po kterém by se mělo zobrazit */usr/bin/ssh*. Pokud se zobrazí chybová hláška *no ssh in*, pak je třeba naistalovat OpenSSH *http://www.openssh.com/*.

Na hlavním uzlu je použit SSHkeygen k vygenerování dvojice RSA klíčů s prázdným heslem. Použití prázdného hesla se nedoporučuje, ale v tomhle případě je potřebné k odemknutí klíče bez zásahu. Jinak by se heslo manuálně zadávalo při každé komunikaci Hadoop s jeho uzly. Vše je prováděno na uživateli **hduser**. [10,15]

```
$ ssh-keygen -t rsa -P ""
```

Dále se v SSH adresáři povolí přístup k lokálnímu počítači s tímto nově vytvořeným klíčem.

```
$ cd ~/.ssh
$ cat id_rsa.pub >> authorized_keys
```

Posledním krokem je otestovat nastavení SSH připojení. Tím se i uživateli **hduser** přidá otisk klíče do souboru known_hosts.

```
$ ssh localhost
```

4.3 Instalace Hadoop s jedním uzlem

Nejprve se z oficiálních stránek Apache Hadoop stáhne balík s aktuální stabilní verzí Hadoop. Stažený archiv se rozbalí a vloží do nově vytvořené složky Hadoop. V tomto případě archiv *hadoop-1.2.1.tar.gz* se rozbalí do */usr/local/Hadoop*. Této složce se musí změnit vlastník všech souborů.

```
$ cd /usr/local
$ sudo tar xzf hadoop-1.2.1.tar.gz
$ sudo mv hadoop-1.2.1 hadoop
$ sudo chown -R hduser:hadoop hadoop
```

Všem uživatelům, kteří chtějí Hadoop používat, je třeba upravit soubor *$HOME/.bashrc*. V tomto souboru se definují konfigurační nastavení prostřednictvím systémových proměnných. Celý obsah, který se přidá na konec tohoto souboru, je v příloze č. 2.

Další konfigurační soubory se nachází ve složce */conf*, celá cesta je */usr/local/hadoop/conf/*. Toto nastavení bude používat HDFS, i když cluster obsahuje jeden lokální počítač. [10]

Jedná se o tyto soubory:

- **hadoop-env.sh**

  Do tohoto souboru se přidá cesta k proměnnému prostředí JAVA_HOME.

  ```
  export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-i386
  ```

- **core-site.xml**

  Obsahuje cestu pro dočasné adresáře a nastavení výchozího souborového systému.

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
</property>
<configuration>
      <property>
            <name>fs.default.name</name>
            <value>hdfs://localhost:54310</value>
      </property>
</configuration>
```

- **hdfs-site.xml**

  Počet replikací může být zadán při vytváření souboru. Pokud není replikace specifikována, tak se použije výchozí hodnota nastavená v tomto souboru.

```
<configuration>
     <property>
         <name>dfs.replication</name>
         <value>1</value>
     </property>
</configuration>
```

- **mapred-site.xml**

  Nastavení hostitele, na kterém běží JobTracker.

```
<configuration>
     <property>
         <name>mapred.job.tracker</name>
         <value>localhost:54311</value>
     </property>
</configuration>
```

Výchozí konfigurace Hadoop využívá *hadoom.tmp.dir* jako základ pro dočasné adresáře a to jak pro lokální systém souborů tak i HDFS. Proto je třeba tento adresář vytvořit a nastavit požadované vlastnictví a oprávnění. HDFS si další důležité podadresáře vytvoří sám, jakmile je bude potřebovat. Umístění adresáře se nastavuje v souboru **core-site.xml**. [5,10,27]

```
$ sudo mkdir -p /app/hadoop/tmp
$ sudo chown hduser:hadoop /app/hadoop/tmp
```

4.3.1 První spuštění

Před samotným spuštěním Hadoop je třeba naformátovat souborový systém. V případě použití formátování opakovaně nebo za chodu se všechna data vymažou.

```
$ /usr/local/hadoop/bin/hadoop namenode –format
```

Příkaz pro spuštění Hadoop a výpis z konzole je na obrázku č. 7.

```
$ /usr/local/hadoop/bin/start-all.sh
```

```
hduser@martin-HP-Compaq-6710b:/usr/local/hadoop$ bin/start-all.sh
Warning: $HADOOP_HOME is deprecated.

starting namenode, logging to /usr/local/hadoop/libexec/../logs/hadoop-hduser-n
amenode-martin-HP-Compaq-6710b.out
localhost: starting datanode, logging to /usr/local/hadoop/libexec/../logs/hado
op-hduser-datanode-martin-HP-Compaq-6710b.out
localhost: starting secondarynamenode, logging to /usr/local/hadoop/libexec/../
logs/hadoop-hduser-secondarynamenode-martin-HP-Compaq-6710b.out
starting jobtracker, logging to /usr/local/hadoop/libexec/../logs/hadoop-hduser
-jobtracker-martin-HP-Compaq-6710b.out
localhost: starting tasktracker, logging to /usr/local/hadoop/libexec/../logs/h
adoop-hduser-tasktracker-martin-HP-Compaq-6710b.out
```

Obr. 7. Spuštění Hadoop služeb

Příkaz pro zastavení Hadoop a výpis z konzole je na obrázku č. 8.

```
$ /usr/local/hadoop/bin/stop-all.sh
```

```
hduser@martin-HP-Compaq-6710b:/usr/local/hadoop$ bin/stop-all.sh
Warning: $HADOOP_HOME is deprecated.

stopping jobtracker
localhost: stopping tasktracker
stopping namenode
localhost: stopping datanode
localhost: stopping secondarynamenode
```

Obr. 8. Zastavení Hadoop služeb

4.3.2 HDFS příkazy

Hadoop poskytuje sadu nástrojů příkazové řádky, které pracují podobně jako příkazy Linux souborů a jsou nezbytné pro interakci s HDFS. Při pracovním postupu se datové soubory vytváří na jiných místech a až následně se kopírují do HDFS. MapReduce programy pak tyto údaje zpracují, ale nečtou žádné HDFS soubory přímo. Místo toho se spoléhají na rámec MapReduce, který načte a zpracuje tyto soubory do jednotlivých záznamů (páry klíč/hodnota). Soubory se čtou přímo jen výjimečně, například při importu a exportu dat. Většina příkazů, které komunikují s clusterem, se provádí pomocí skriptů *bin/hadoop*. Ten načte systém Hadoop spolu s JVM a spustí příkazy uživatele.

Příkazy pro HDFS mají formu: *hadoop$ bin/hadoop modul –cmd <args>*.

Modul řekne programu, kterou podmnožinu funkcí Hadoop používat, dále *cmd* je název specifického příkazu v rámci modulu a *<args>* jsou argumenty. Než se začnou zadávat příkazy, předpokládá se, že je HDFS správně naformátován a spuštěn souborový systém. [10,15,27]

Existují dva moduly *dfs* a *dfsadmin*. DFS modul poskytuje základní příkazy pro správu souborů, které pracují s objekty v rámci souborového systému.

Základní příkazy jsou:

- Vytvoření složky: *bin/hadoop dfs -mkdir /user/hduser*
- Výpis složek: *bin/hadoop dfs -ls*
- Vkládání souborů: *bin/hadoop dfs -put file.txt /user/hduser* nebo *-copyFromLocal*
- Získání souborů: *bin/hadoop dfs -get file.txt*
- Mazání souborů: *bin/hadoop dfs -rm file.txt*
- Více informací: *bin/hadoop dfs -help*

Dfsadmin modul se dotazuje nebo manipuluje se souborovým systémem jako celek.

- Získaní celkového stavu clusteru: *dfsadmin –report*, ukázka v příloze č. 3.
- Získání stavu Namenode: *bin/hadoop dfsadmin -metasave file*
- Nouzový režim: *bin/hadoop dfsadmin –safemode enter* / *leave*
- Více informací: *bin/hadoop dfsadmin –help*

Snadnější způsob, jak manipulovat se soubory v HDFS pomocí Eclipse pluginu je popsán v kapitole 4.5.

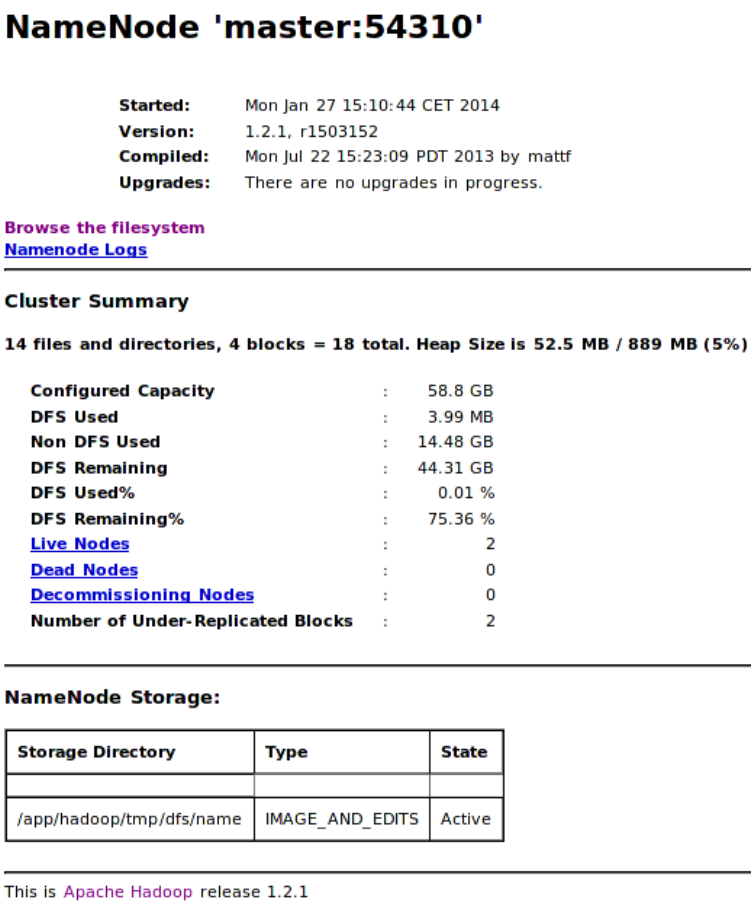
4.3.3 Webové rozhraní

Hadoop nabízí několik webových rozhraní, které poskytují informace o tom, co se na clusteru děje. Jejich umístění se nastavuje v souboru *conf/hadoop-default.xml* a jedná se o:

- Webové rozhraní Namenode viz obr. č. 9.

  Zobrazuje informace o aktuálním stavu clusteru, včetně informací o kapacitě nebo uzlech. Navíc umožňuje procházet jmenný prostor HDFS a zobrazit obsah souborů přímo v prohlížeči. Obsahuje i Namenode log soubory.

  Ve výchozím nastavení je k dispozici na *http://localhost:50070/*.

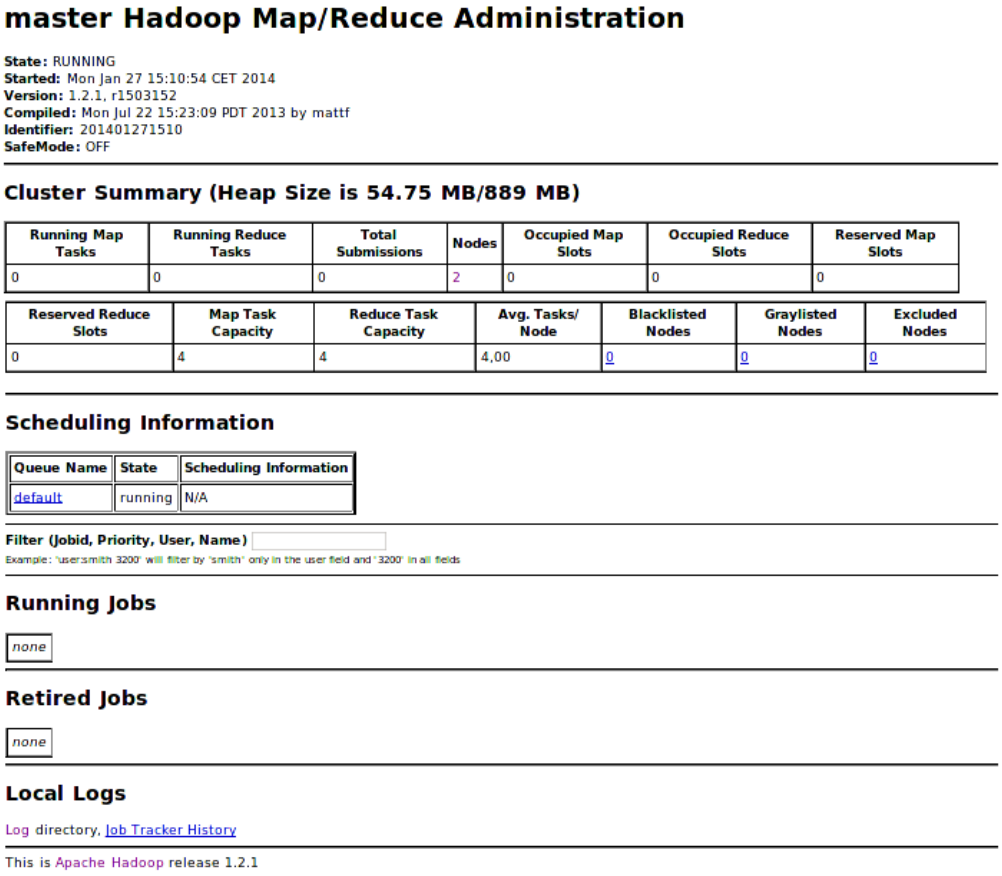


Obr. 9. Webové rozhraní Namenode

- Webové rozhraní JobTracker viz obr. č. 10.

  Poskytuje všeobecné informace a statistiky o úkolech na Hadoop clusteru. Obsahuje informace o plánování nebo informace o běžících, dokončených a přerušených úkolech.

  Ve výchozím nastavení je k dispozici na *http://localhost:50030/*.



master Hadoop Map/Reduce Administration

State: RUNNING
Started: Mon Jan 27 15:10:54 CET 2014
Version: 1.2.1, r1503152
Compiled: Mon Jul 22 15:23:09 PDT 2013 by mattf
Identifier: 201401271510
SafeMode: OFF

Cluster Summary (Heap Size is 54.75 MB/889 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots
0	0	0	2	0	0	0

Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/ Node	Blacklisted Nodes	Graylisted Nodes	Excluded Nodes
0	4	4	4,00	0	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

none

Retired Jobs

none

Local Logs

Log directory, Job Tracker History

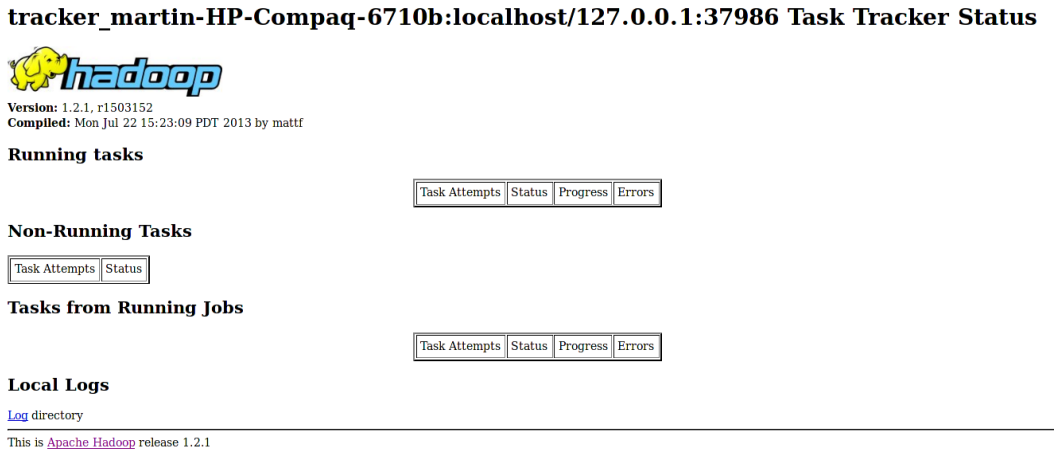
This is Apache Hadoop release 1.2.1

Obr. 10. Webové rozhraní JobTracker

- Webové rozhraní TaskTracker viz obr. č. 11.

  Zobrazuje informace o běžících a neběžících úkolech. Jejich stavu, průběhu nebo chybách.

  Ve výchozím nastavení je k dispozici na *http://localhost:50060/*.



tracker_martin-HP-Compaq-6710b:localhost/127.0.0.1:37986 Task Tracker Status

Version: 1.2.1, r1503152
Compiled: Mon Jul 22 15:23:09 PDT 2013 by mattf

Running tasks

Task Attempts	Status	Progress	Errors

Non-Running Tasks

Task Attempts	Status

Tasks from Running Jobs

Task Attempts	Status	Progress	Errors

Local Logs

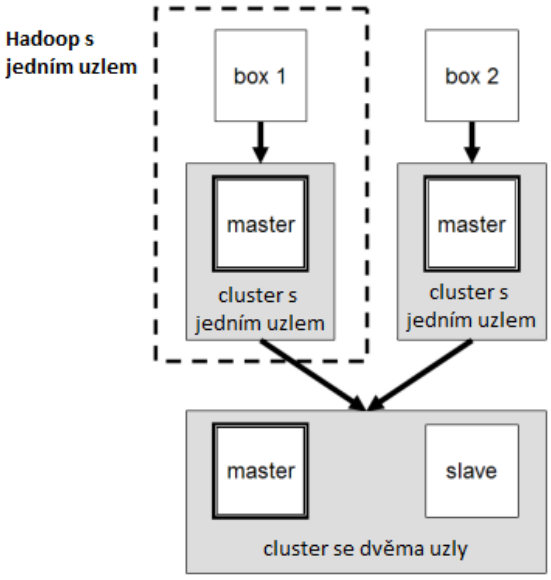
Log directory

This is Apache Hadoop release 1.2.1

Obr. 11. Webové rozhraní TaskTracker

4.4 Nastavení clusteru s více uzly

Pokud chceme, aby Hadoop běžel v clusteru, provedeme na každém počítači instalaci varianty s jedním uzlem popsanou v kapitole 4.3 a spuštěním ověříme funkčnost. Pokud je všechno v pořádku, tak se můžou upravit konfigurační soubory pro zprovoznění clusteru s více uzly jako na obr. č. 12.



Obr. 12. Struktura clusteru s dvěma uzly [11]

Nejprve se nastaví počítače tak, aby byly navzájem dosažitelné v síti, a upraví se jejich soubor etc/hosts přidáním jejich IP adres.

- **etc/hosts**

  192.168.0.1 master

  192.168.0.2 slave

Taky se musí vložit veřejný SSH klíč master počítače do autorizovaných klíčů slave počítače. To se provede příkazem:

```
$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@slave
```

Správné SSH připojení se může ověřit připojením z master do slave počítače příkazem *ssh slave*. Tím se přidá otisk slave klíče do souboru known_hosts master počítače.

Jelikož je na každém počítači varianta s jedním uzlem, musí se změnit i konfigurační soubory */conf/*-site.xml* na všech strojích. Hadoop má velkou řadu konfiguračních vlastností, které je třeba definovat pro každý cluster reálného světa.

V tabulce číslo 3 jsou vypsány základní vlastnosti souborů *core-site.xml* a *hdfs-site.xml*.

Název vlastnosti	Typ	Výchozí hodnota	Popis
fs.default.name	URI	file://	Výchozí souborový systém. URI definuje název hostitele a port, na kterém běží Namenode server.
dfs.name.dir	Názvy adresářů oddělené čárkami	${hadoop.tmp.dir}/ dfs/name	Seznam adresářů, kde namenode ukládá své trvalé metadata.
dfs.data.dir	Názvy adresářů oddělené čárkami	${hadoop.tmp.dir}/ dfs/data	Seznam adresářů, kde datanode ukládá bloky. Každý blok je uložen pouze v jednom z těchto adresářů.
fs.checkpoint.dir	Názvy adresářů oddělené čárkami	${hadoop.tmp.dir}/ dfs/namesecondary	Seznam adresářů, kde sekundární namenode ukládá kontrolní body.

Tab. 3. Podstatné HDFS vlastnosti [19]

V následující tabulce číslo 4 jsou vypsány základní vlastnosti souboru *mapred-site.xml*.

Název vlastnosti	Typ	Výchozí hodnota	Popis
mapred.job.tracker	Hostname a port	local	Název hostitele a port, na kterém běží JobTracker server.
mapred.local.dir	Názvy adresářů oddělené čárkami	${hadoop.tmp.dir} /mapred/local	Seznam adresářů, kde MapReduce ukládá přechodné data. Po dokončení úkolu jsou smazána.
mapred.system.dir	URI	${hadoop.tmp.dir} /mapred/system	Relativní adresář k fs.default.name, kde jsou uložené sdílené soubory během běhu souboru.
mapred.tasktracker. map.tasks.maximum	int	2	Počet map úkolů, které mohou být spuštěny na tasktracker najednou.
mapred.tasktracker. reduce.tasks.maximum	int	2	Počet reduce úkolů, které mohou být spuštěny na tasktracker najednou.

Tab. 4. Podstatné MapReduce vlastnosti [19]

Ukázka nastavení souborů.

- **core-site.xml**

```
<name>fs.default.name</name>
<value>hdfs://master:54310</value>
<final>true</final>
```

- **mapred-site.xml**

```
<name>mapred.job.tracker</name>
<value>master:54311</value>
<final>true</final>
```

- **hdfs-site.xml**

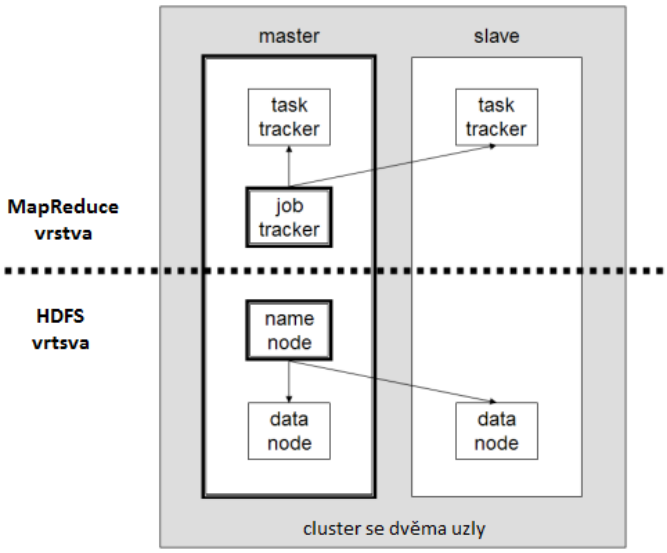
```
<name>dfs.replication</name>
<value>2</value>
```

Označením vlastností jako *final* se zabrání jejich přepsání z konfigurace úkolu.

4.4.1 Spuštění clusteru

Cluster se spustí stejně, jako v případě s jedním uzlem (*bin/start-all.sh*) z počítače označeného jako **master**. Popřípadě se spuštění může rozdělit na dva kroky, tak že se nejprve spustí HDFS démoni (*bin/start-dfs.sh*) a poté MapReduce démoni (*bin/start-mapred.sh)*. Tohle rozdělení je výhodnější v případě ladění, protože každý démon běží ve svém terminále. Příkazem `$jps` se může vypsat status procesů JVM. Na jednotlivých počítačích by měly běžet procesy jako na obrázku č. 13.

- Master – tasktracker, jobtracker, namenode, datanode, secondary namenode
- Slave – tasktracker, datanode



Obr. 13. Výsledný cluster se dvěma uzly [11]

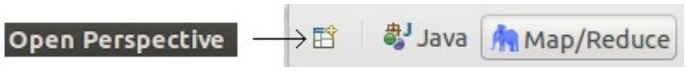
4.5 Eclipse Plugin

Eclipse je open source vývojová platforma, určená pro programování v jazyce Java. Flexibilní návrh této platformy dovoluje rozšířit seznam podporovaných programovacích jazyků za pomoci pluginů. Hadoop Eclipse Plugin poskytuje nástroje pro snadnou práci s MapReduce. Pro vytvoření pluginu jsou zapotřebí zdrojové soubory Hadoop. Ty se můžou například získat z SVN uložiště pomocí Subclipse ze stránek *http://svn.apache.org/repos/asf/hadoop/common/tags/*. Vybereme svou verzi Hadoop a stáhneme zdrojové soubory. Tím se vytvoří nový projekt MapReduceTools. Tento projekt se musí upravit. Podrobný návod pro Hadoop verzi 1.2.1 je například na stránkách *http://ordinarygeek.me/2013/09/12/hadoop-build-and-install-hadoop-1-2-1-eclipse-plugin/*.

Sestavení pluginu do adresáře *{hadoop-src-root}/build/contrib/eclipse-plugin/hadoop-${version}-eclipse-plugin.jar* se provede v Eclipse, pravým kliknutím na soubor build.xml a vybere se možnost *Run As -> Ant build*. [25]

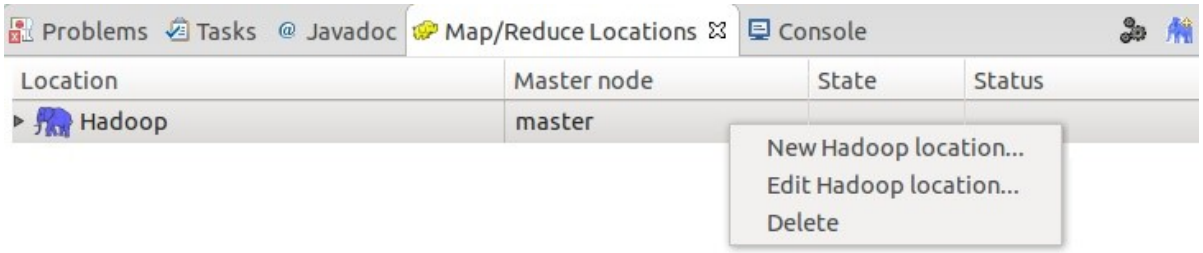
4.5.1 Konfigurace pluginu

Vytvořený plugin se umístí do složky */eclipse/plugins*. To umožní přepnutí Eclipse do Map/Reduce perspektivy.



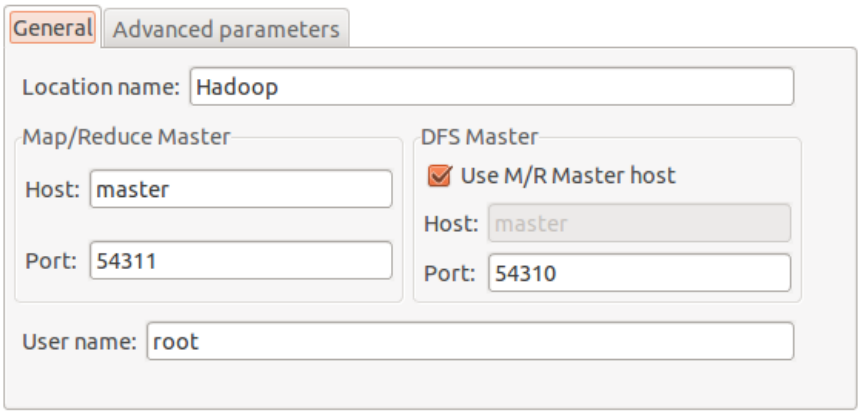
Obr. 14. MapReduce pohled

V tomto pohledu se vedle konzole zobrazí nový panel s názvem Map/Reduce Locations, kde se vytváří Hadoop připojení.



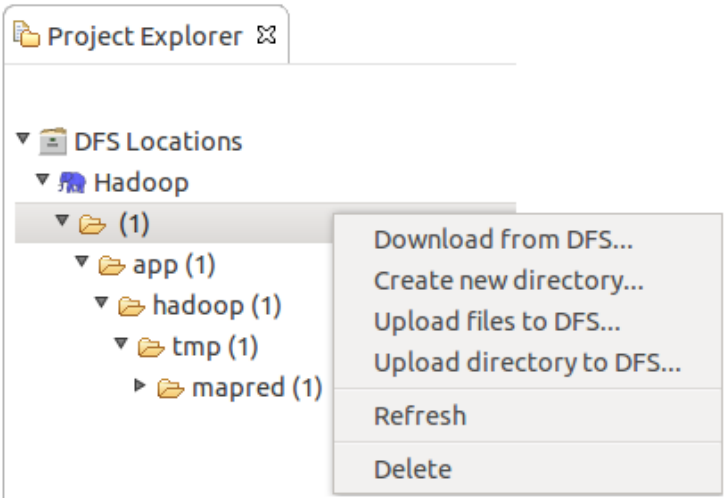
Obr. 15. Panel Map/Reduce Locations

Pro přidání připojení se pomocí *New Hadoop location* zobrazí okno pro definování nového Hadoop umístění. V tomto okně se definuje libovolný název připojení, host a porty nastavené v souborech *core-site.xml* a *mapred-site.xml*. Pro pokročilé nastavení slouží druhá záložka *Advanced parameters*.



Obr. 16. Definice Hadoop umístění

Pokud je Hadoop spuštěn a je nastaven plugin, zobrazí se HDFS umístění mezi projekty. To usnadňuje celkovou práci a není třeba provádět příkazy v příkazovém řádku.



Obr. 17. Prohlížení a práce s HDFS

Mezi hlavní přednosti pluginu patří:

- Vytváření Map, Reduce a Drive třídy.
- Prohlížení a práce s distribuovaným souborovým systémem (Obr. č. 17).
- Předložení a sledování provádění úloh.

5 IMPLEMENTACE MAPREDUCE

Tato kapitola se zabývá způsobem implementace MapReduce na různých problémech. Implementace je prováděna jak pomocí Hadoop, tak i vlastní. Poté je na těchto implementacích otestována efektivita řešení pro různé dimenze dat. Veškeré zdrojové kódy jsou v příloze na CD.

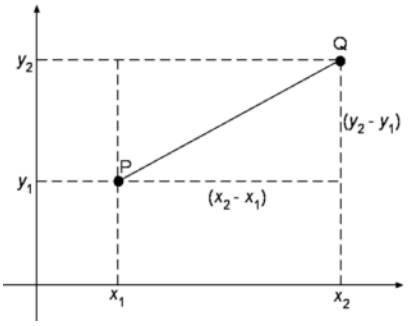
Implementace a testování je prováděno na těchto problémech:

- Výpočet Euklidovské matice vzdáleností pro velký počet bodů

  Euklidovská matice vzdáleností je matice A o velikosti N x N představující vzdálenosti množiny N bodů v Eukleidovském prostoru. Prvek matice a_{ij} popisuje vzdálenost mezi i-tým a j-tým bodem z množiny. Matice má následující vlastnosti:

  - Všechny prvky na diagonále A jsou nulové.
  - Součet prvků na hlavní diagonále je 0 (podle výše uvedené vlastnosti).
  - A je symetrická, tedy $a_{ij} = a_{ji}$.
  - $a_{ij} \geq 0$

  Euklidovská vzdálenost mezi body **P** a **Q** je délka úsečky, která tyto body spojuje. Jestliže jsou dány body **P** = (x_1 , y_1) a **Q** = (x_2 , y_2). Jejich vzdálenost se vypočítá pomocí vztahu $d(\mathbf{P}, \mathbf{Q}) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. Tato vzdálenost je vypočítána pro všechny body a vkládána do matice A_{pq}. [20]



Obr. 18. Vzdálenost mezi dvěma body

- Počítání výskytu slov

  Jedná se o jednoduchý příklad tzv. WordCount, který spočítá počet výskytů každého slova ve vstupních souborech. Při počítání se nebude rozlišovat velikost písmen.

Při práci je používána následující verze OS a softwaru:

- Ubuntu verze 13.10 „Saucy Salamander“
- Hadoop 1.2.1 stable, vydáno 1. 8. 2013
- Java 1.7.0_25
- Eclipse Kepler Service Release 2

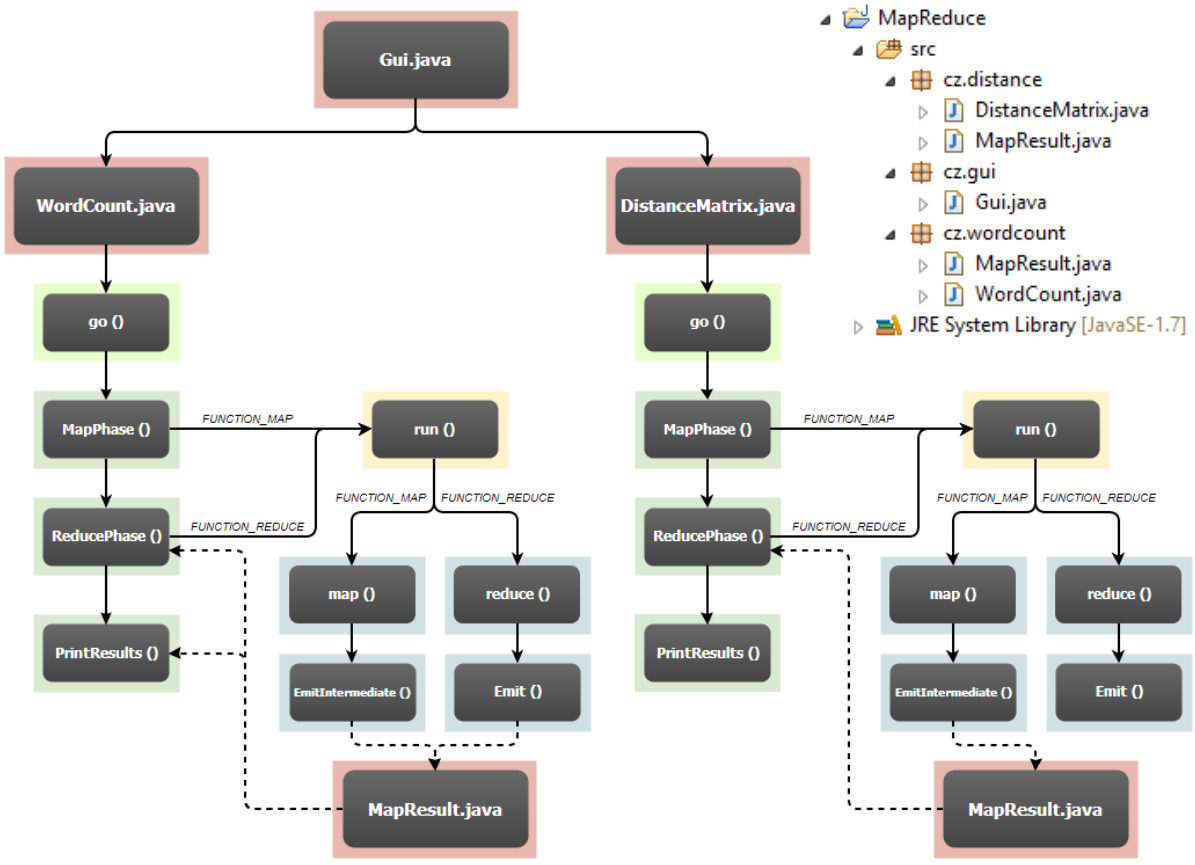
Nejprve je popsán způsob vlastní implementace a poté implementace pomocí Hadoop.

5.1 Vlastní implementace

Vlastní implementaci je prováděna ve vývojovém prostředí Eclipse a kód je napsán v jazyce Java. Neexistuje žádná formální definice MapReduce modelu, proto je zvolen průběh zpracování v obou implementacích následovně:

- Vstupní soubory jsou rozděleny do více Map procesů, které se vykonávají souběžně. Každý pracovník (vlákno) dostane vlastní soubor ke zpracování. Pokud je zadán pouze jeden vstupní soubor, pak se vytvoří pouze jeden pracovník.
- Dočasné páry klíč/hodnota jsou pomocí metody EmitIntermediate uloženy pro fázi Reduce.
- V Reduce fázi jsou dočasné páry rozděleny do bloků podle počtu Reduce pracovníků a poté těmto pracovníkům přiděleny. Počet pracovníku se určí před výběrem vstupních souborů.
- Výsledky Reduce fáze se uloží a je zavolána metoda PrintResults, která tyto výsledky zapíše do souboru.

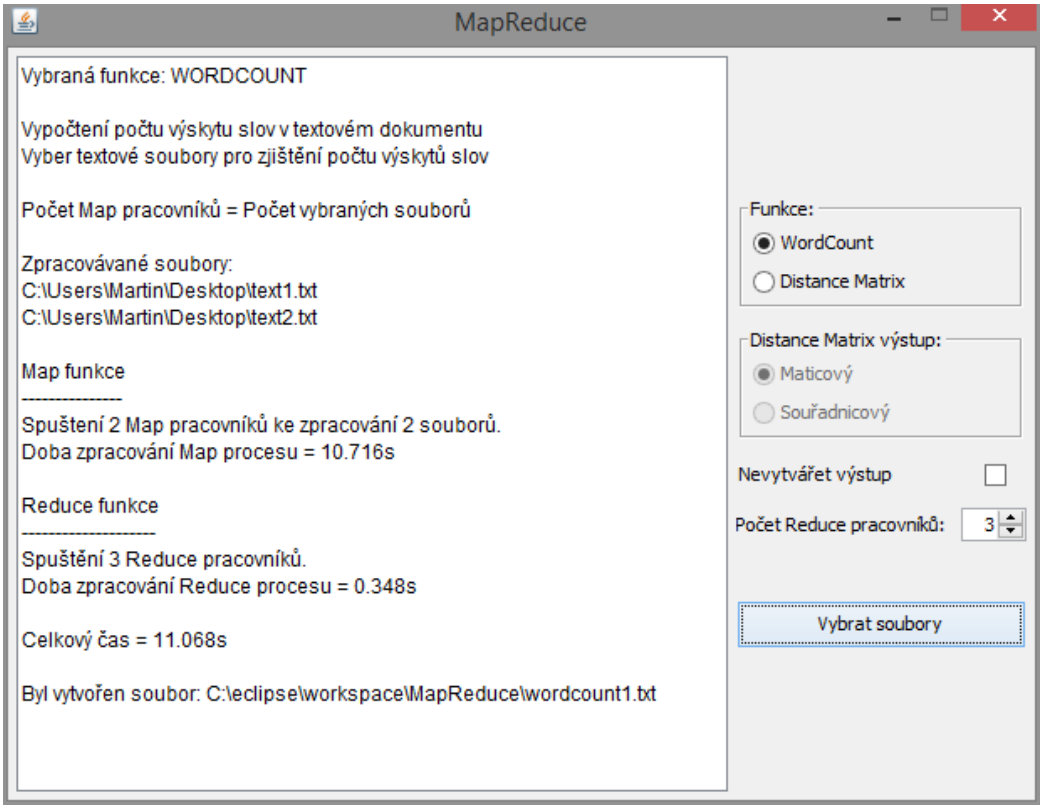
Na obrázku č. 19 je zobrazeno složení tříd a postup metodami v těchto třídách.



Obr. 19. Schéma aplikace a její třídy

5.1.1 Grafické rozhraní

Pro vlastní implementaci bylo vytvořeno grafické rozhraní. Toto rozhraní umožňuje snadné testování a zobrazuje důležité informace o průběhu a výsledcích úkolu. Je zobrazeno na obrázku č. 20.



Obr. 20. Grafické rozhraní

Grafické rozhrání obsahuje:

- Textová oblast – Zobrazuje informace před a po provedení úkolu. Pokud je prováděno více úkolů stejné funkce po sobě, pak informace o předešlých úkolech zůstávají vypsány.
  Při provedení úkolu se zobrazí:
  - Úplné cesty ke zpracovávaným souborům
  - Počet spuštěných Map pracovníků
  - Čas provádění Map procesu
  - Počet spuštěných Reduce pracovníků
  - Čas provádění Reduce procesu
  - Celkový čas (počítáno bez vytváření výsledného souboru)
  - Úplná cesta k vytvořenému souboru s výsledky
- Výběr prováděného úkolu (Funkce) – Na výběr jsou dvě možnosti WordCount nebo Distance Matrix. Podle toho, která možnost je vybrána se v textové oblasti zobrazí informace k úkolu.

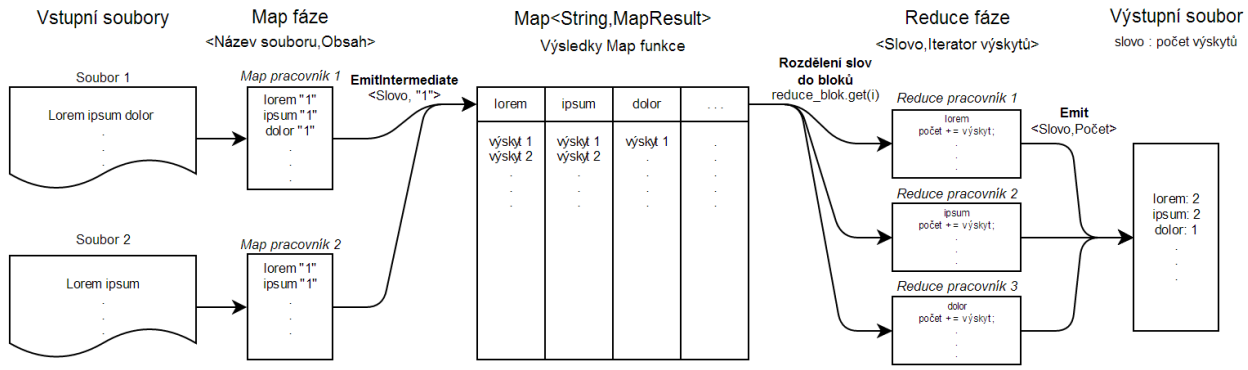
- Distance matrix výstup – Volba je aktivní pouze u funkce matice vzdáleností a jedná se o způsob zápisu výsledků do konečného textového souboru. Maticový výstup vytvoří přímo matici vzdáleností, zatímco souřadnicový výstup vypíše souřadnice matice, které označují jednotlivé body a vzdálenost mezi nimi. Tato volba nemá žádný vliv na zpracování úkolu. Slouží pouze pro snadnější srovnání výsledků s Hadoop výsledky.
- Nevytvářet výstup – Výstupní soubor slouží především ke kontrole správného provedení úkolu a k testování efektivity řešení není vůbec zapotřebí. Taky jeho vytváření může trvat dlouho dobu v případě velkých vstupních dat. Proto je zde umístěna volba nevytvářet výstup, kdy se provede pouze Map a Reduce funkce.
- Počet Reduce pracovníků – Nastavení, kolik Reduce pracovníků bude zpracovávat přechodné páry z funkce Map. Volba se provádí před výběrem souborů.
- Výběr vstupních souborů – Jedná se o tlačítko, které zobrazí dialogové okno pro výběr souborů. Dialogu je nastaven filtr, aby pracoval pouze s textovými soubory. Jakmile uživatel označí soubory a klikne na Open, tak se začne provádět úloha podle dříve vybrané funkce. Vstupní soubory se pro každou funkci liší. Při vybrané funkci Distance Matrix je prováděna jednoduchá kontrola formátu vstupních souřadnic. Pokud jsou souřadnice v nesprávném formátu, pak se zobrazí chybová hláška s názvem souboru, ve kterém se vyskytl problém a aplikace se ukončí.

Po úspěšném provedení úkolu se ve stejné složce, ve které je spuštěná aplikace, vytvoří textový soubor s výsledky. Jeho název je dán prováděnou funkcí a pořadovým číslem a to pro případ, že již daný soubor existuje. Například wordcount1.txt, wordcount2.txt, dMatrix1.txt, atd. Principiálně jsou obě úlohy podobné. Liší se samozřejmě v map() a reduce() metodě, ale i v ukládání výsledků Map funkce a ve způsobu rozdělování těchto výsledků do bloků pro Reduce pracovníky.

Jelikož třídy WordCount i DistanceMatrix jsou základ pro Map i Reduce pracovní vlákna, tak implementují rozhraní java.lang.Runnable. Proto obsahují metodu run (), která pomocí Switche umožňuje na základě hodnoty proměnné přepínat mezi funkcemi Map a Reduce.

5.1.2 Výpočet výskytu slov

V této části jsou popsány jednotlivé operace výpočtu výskytu slov, jak následují po sobě. Na obrázku č. 21 je zobrazen průběh zpracování pro 2 Map pracovníky a 3 Reduce pracovníky.



Obr. 21. Průběh vlastní implementace WordCount

Třída WordCount

Vstupní soubory můžou být jakékoliv textové soubory, které obsahují nějaký text. Při zvolení funkce WordCount v grafickém rozhraní a vybrání vstupních souborů se zavolá třída *WordCount.java* a vybrané soubory se uloží do seznamu *files*.

Metoda go ()

Obsahuje volání map fáze, reduce fáze, výpočet celkového času a tisk výsledků.

Metoda MapPhase ()

Map fáze vytvoří pro každý vstupní soubor pracovníka. Pomocí konstruktoru se určí, který soubor má daný pracovník zpracovávat a kterou funkci provádět.

```
WordCount worker = new WordCount(FUNCTION_MAP, files[i]);
```

Pro výsledky této fáze je vytvořena synchronizovaná HashMapa, ve které jsou uloženy všechny souřadnice.

```
map_results=Collections.synchronizedMap(new HashMap<String,MapResult>());
```

Po spuštění vláken se program přesune do metody *run ()*.

Metoda run () *Map funkce*

V této části se pomocí metody *GetFileContents()* načte obsah daného souboru. Obsah je načítán do StringBufferu. Je podobný jako String, ale může být modifikován a je bezpečný při použití více vláken. V případě potřeby můžou být metody synchronizovány. Jelikož interně používá pole (char []), tak maximální počet prvků, které může pojmout je $2^{31} - 1$ (Integer.MAX_VALUE).

```
String line = reader.readLine();
while (line != null) {
	contents.append(line.toLowerCase());
	contents.append(" ");
	line = reader.readLine();
}
```

Všechny znaky v řetězci se převádí na malá písmena a při připojení nového řádku se vloží mezera. Jinak by všechny slova na konci a začátku řádku byly spojeny.

Poté je volána metoda map (), které se předává název a obsah souboru.

```
map(file.getName(), contents);
```

Metoda map ()

Nejprve se rozdělí řetězec na pole jednotlivých slov. Toho se dosáhne pomocí předdefinované třídy znaků, která nabízí zkrácené zápisy pro běžně používané regulární výrazy.

```
String[] words = value.split("\\W");
```

Zkratka \\W znamená znak, které není slovo. Tedy negovaný slovní znak [a-zA-Z_0-9]. Projde se celé pole slov a každé je posíláno spolu s číslem jedna do metody EmitIntermediate ().

```
for (String word : words) {
	EmitIntermediate(word, "1");
```

Metoda EmitIntermediate ()

Nyní se páry (slovo, “1“) nachází v poslední části map fáze. Nejprve se zkontroluje, zda ve výsledcích map_results existuje záznam o aktuálním klíči. Pokud ne, tak se vytvoří nový a hodnota je přidána do seznamu výsledků. Jinak je hodnota přidána do seznamu k záznamu se stejným klíčem. Taky je použit zámek na map_results, aby nedošlo k chybě při souběžném vkládání více vláken najednou.

```
synchronized (map_results) {
	group = map_results.get(key);
	if (group == null) {
		group = new MapResult(key);
		map_results.put(key, group);
	}
}
group.addValue(value);
```

Metoda ReducePhase ()

V Reduce fázi se z grafického rozhraní načte počet pracovníků, kteří mají zpracovat výsledky map fáze. Vypočítá se velikost bloku pro každého pracovníka a celkový počet záznamů. Sada MapResult objektů z tabulky map_results je rozdělena do bloků podle počtu Reduce pracovníků. Prochází se jednotlivé záznamy a ty jsou vkládány do bloků. Jakmile je blok plný, odečtou se již vložené záznamy a zbylé se vkládají do dalšího bloku. Tohle se opakuje, dokud jsou záznamy, které by se dali odečíst

z celkového počtu záznamů. Pro poslední blok se v podmínce *if* přiřadí přesný počet zbývajících záznamů. Každý pracovník dostane všechny hodnoty pro klíč, protože jsou uloženy v jediném MapResult objektu.

```
while (records_remaining > 0) {
      if (records_remaining < block_size) {
            block_size = records_remaining;
      }
      worker_map = new HashMap<String, MapResult>(block_size);
      reduce_chunks.add(worker_map);
      for (int record = 0; record < block_size; record++) {
            MapResult m = iterator.next();
            worker_map.put(m.getKey(), m);
      }
      records_remaining -= block_size;
}
```

Každému pracovníkovi se přiřadí proměnná pro přechod do funkce Reduce v metodě run () a blok se záznamy.

```
WordCount worker = new WordCount(FUNCTION_REDUCE,reduce_chunks.get(i));
```

Metoda run () *Reduce funkce*

Po spuštění vláken se opět program dostává do metody run (), ale nyní mají vlákna přiřazenou funkci Reduce. Z bloku se záznamy je vytvořena sada, která se prochází pomocí iterátoru. Pro každý záznam je volána metoda reduce (). Je zde použita proměnná current_result, která je členem objektu MapResult.

```
while (results_iterator.hasNext()) {
      Map.Entry<String, MapResult> result_entry =
      results_iterator.next();
      current_result = result_entry.getValue();
      reduce(current_result.key, current_result.iterator());
}
```

Metoda reduce ()

V této metodě se provede sečtení všech výskytů pro jednotlivá slova. A ta se předá metodě Emit().

```
while (values.hasNext()) {
      result += Integer.parseInt(values.next());
}
Emit(result + "");
```

Metoda Emit ()

Pomocí proměnné current_result se přiřadí emitovaný výsledek k MapResult objektu.

```
current_result.setResult(result);
```

Metoda PrintResult ()

Metoda vytvoří výsledný textový soubor s počty výskytů jednotlivých slov.

```
BufferedWriter output = new BufferedWriter(new FileWriter(file));
            while (iterator.hasNext()) {
                  MapResult result = iterator.next();
                  output.write(result.getKey()+":"+result.getResult());
                  output.newLine();
            }
            output.close();
```

Třída MapResult

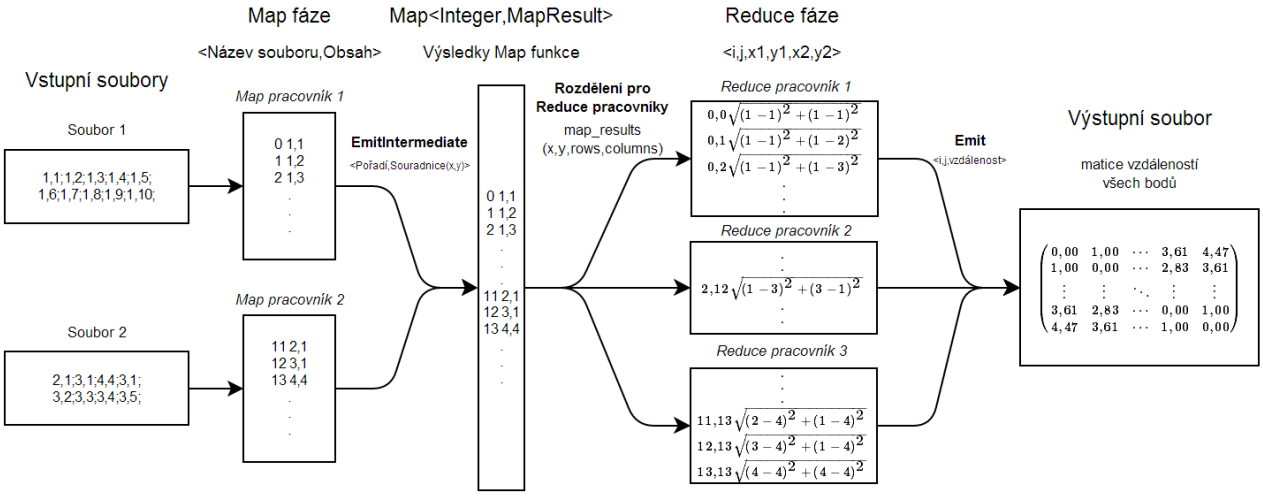
Jedná se o pomocnou třídu, která slouží hlavně k propojení konkrétního klíče se seznamem přechodných hodnot z map fáze a poté s konečným výsledkem z Reduce fáze. Jakmile je zavolána metoda EmitIntermediate (klíč, hodnota) s novým klíčem, pak je pro něj vytvořen LinkedList a do tohoto listu se ukládají všechny hodnoty se stejným klíčem.

```
public MapResult(String _key) {
      key = _key;
      values = Collections.synchronizedList(new LinkedList<String>());
}
```

Dále obsahuje metody pro ukládání nových hodnot addValue(), uložení a získání výsledku setResult() a getResult() nebo metodu pro procházení hodnot iterator().

5.1.3 Matice vzdálenosti

Stejně jako v předchozím případě je na obrázku č. 22 zobrazen ukázkový průběh zpracování vstupních souborů pro výpočet matice vzdáleností. Jsou použiti 2 Map pracovníci a 3 Reduce pracovníci.



Obr. 22. Průběh vlastní implementace matice vzdáleností

Třída DistanceMatrix

Souřadnice ve vstupních souborech pro tuto úlohu jsou odděleny čárkou a celé body jsou odděleny středníkem. Musí být tedy ve formátu **x1,y1;x2,y2;** atd. Práce třídy DistanceMatrix začíná, jakmile je v grafickém rozhraní zvolena možnost DistanceMatrix a jsou vybrány vstupní soubory. Ty se uloží do seznamu souborů s názvem *files* a zavolá se metoda go ().

Metoda go ()

Obsahuje volání metod *MapPhase ()*, *ReducePhase ()*, *PrintResults ()*. Zůstává tedy stejná jako v předchozím případě.

Metoda MapPhase ()

Map fáze vytvoří pro každý vstupní soubor pracovníka. Pomocí konstruktoru se určí, který soubor má daný pracovník zpracovávat a kterou funkci provádět.

```
DistanceMatrix worker = new DistanceMatrix(FUNCTION_MAP, files[i]);
```

Pro výsledky této fáze je opět vytvořena synchronizovaná HashMapa, ve které jsou uloženy všechny souřadnice.

```
map_results = Collections.synchronizedMap(new HashMap<Integer, MapResult>());
```

Jakmile se spustí vlákna, JVM zavolá metodu run ().

Metoda run () *Map funkce*

V této části pomocí metody GetFileContents() je načten obsah daného souboru. Pokud jsou souřadnice zadány ve více řádcích, tak jsou spojeny do jednoho.

```
while (line != null) {
      contents.append(line);
      line = reader.readLine();
}
```

Pak je zavolána metoda map (), které se předává název a obsah celého souboru.

```
map(file.getName(), contents);
```

Metoda map ()

Nejprve se rozdělí celý obsah souboru na jednotlivé body:

```
String[] words = value.split(";");
```

Poté se zkontroluje formát bodů a rozdělí se na jednotlivé souřadnice. Tyto souřadnice jsou předány metodě EmitIntermediate spolu s jejich pořadím.

```
for (String word : words) {
      if (word.matches("^[0-9]+,[0-9]+$")) {
            String[] split = word.split(",");
            EmitIntermediate(count,   Integer.parseInt(split[0]),
                                      Integer.parseInt(split[1]));

      }
}
```

Metoda EmitIntermediate ()

Jedná se o poslední část Map fáze. Ke každému klíčí je ve třídě MapResult vytvořeno pole pro souřadnice x a y. Každý klíč s tímto polem je uložen do hashmapy *map_results*.

```
coordinates = map_results.get(key);
      if (coordinates == null) {
            coordinates = new MapResult(key);
            map_results.put(key, coordinates);
      }
coordinates.addValue(x, y);
```

Metoda ReducePhase ()

V Reduce fázi se z grafického rozhraní načte počet pracovníků, kteří mají zpracovat výsledky map fáze. Každému je přiřazena část výsledné matice, kterou mají zpracovat, data potřebné k výpočtu a funkci, kterou mají provádět.

```
DistanceMatrix worker = new DistancMatrix
(FUNCTION_REDUCE,reduce_chunks.get(i),0,part_y[i],rows,column[i]);
```

Způsob rozdělení matice na části je zobrazen na obrázku č. 23.

Vstupní data:
1,1;1,2;1,3;1,4;1,5;1,6;1,7;1,8;1,9;1,10
Počet bodů: 10
Počet Reduce pracovníků: 3

0	1	2	3	4	5	6	7	8	9
1	0	1	2	3	4	5	6	7	8
2	1	0	1	2	3	4	5	6	7
3	2	1	0	1	2	3	4	5	6
4	3	2	1	0	1	2	3	4	5
5	4	3	2	1	0	1	2	3	4
6	5	4	3	2	1	0	1	2	3
7	6	5	4	3	2	1	0	1	2
8	7	6	5	4	3	2	1	0	1
9	8	7	6	5	4	3	2	1	0

Vlákno	x	y	řádků	sloupců
0	0	0	10	4
1	0	4	10	3
2	0	7	10	3

Vstupní data:
1,1;1,2;1,3;1,4;1,5;1,6;1,7;1,8;1,9;1,10;1,11
Počet bodů: 11
Počet Reduce pracovníků: 3

0	1	2	3	4	5	6	7	8	9	10
1	0	1	2	3	4	5	6	7	8	9
2	1	0	1	2	3	4	5	6	7	8
3	2	1	0	1	2	3	4	5	6	7
4	3	2	1	0	1	2	3	4	5	6
5	4	3	2	1	0	1	2	3	4	5
6	5	4	3	2	1	0	1	2	3	4
7	6	5	4	3	2	1	0	1	2	3
8	7	6	5	4	3	2	1	0	1	2
9	8	7	6	5	4	3	2	1	0	1
10	9	8	7	6	5	4	3	2	1	0

Vlákno	x	y	řádků	sloupců
0	0	0	11	4
1	0	4	11	4
2	0	8	11	3

Obr. 23. Ukázka rozdělení práce pro Reduce pracovníky

Rozdělení spočívá ve vypočtení počátečního místa v matici, odkud má dané vlákno začít pracovat a počtu řádků a sloupců, které má zpracovat. O tuto práci se stará následující část kódu:

```
int div = block_size/num_reduce_workers;
int div2 = block_size%num_reduce_workers;
for (int i = 0; i < column.length; i++) {
	part_y[i] = y;
	if (div2 > 0) {
		column[i] = div + 1;
		--div2;
	} else {
		column[i] = div;
	}
	y = y + column[i];
}
```

Metoda run () *Reduce funkce*

Po spuštění vláken se opět kód dostává do metody run (), ale nyní má přiřazenou funkci Reduce. Každému vláknu jsou předány souřadnice všech bodů, protože se potřebují vypočítat všechny řádky matice. O průchod dané části matice se starají dva speciální vnořené cykly for.

```
for (int i = x; i < x + rows; i++) {
	for (int j = y; j < y + columns; j++) {
```

V cyklech for je volána metoda reduce (), které se předává souřadnice dvou aktuálních bodů a souřadnice, které určí umístění vzdálenosti těchto bodů ve výsledné matici tedy i, j, x1, y1, x2, y2.

Metoda reduce ()

V této metodě se provede výpočet požadované vzdálenosti dvou bodů. A ta se předá metodě Emit().

```
distance = Math.pow(Math.pow((x1-x2), 2.0) + Math.pow((y1-y2), 2.0), 0.5);
Emit(i, j, distance);
```

Metoda Emit ()

Metoda Emit() je poslední část Reduce fáze a taky celého MapReduce úkolu. Provádí uložení vzdálenosti do dvourozměrného pole.

```
private void Emit(Integer x, Integer y, double matrix2) {
	matrix[x][y] = matrix2;
```

Po dokončení výpočtu všech vzdáleností se program opět přesune do metody go (). V té se ukončí počítání celkového času úlohy a přejde se na tisk výsledků do souboru.

Metoda PrintResult ()

Jedná se o poslední část programu, ve kterém se vytvoří nový soubor s názvem dMatrix a pořadovým číslem.

```
int p = 2;
File file = new File("dMatrix1.txt");
	while (file.exists()) {
		file = new File("dMatrix" + p + ".txt");
		p++;
	}
```

A v tomto souboru je uložena výsledná matice vzdáleností.

```
BufferedWriter output = new BufferedWriter(new FileWriter(file));
for (int i = 0; i < matrix.length; i++) {
	for (int j = 0; j < matrix[i].length; j++) {
		String format = String.format("%5.2f\t",matrix[i][j]);
		output.write(format);
		}
	output.newLine();
}
output.close();
```

Třída MapResult

Jedná se o pomocnou třídu, která slouží hlavně k propojení konkrétního klíče s jeho souřadnicemi. Jakmile je zavolána metoda EmitIntermediate (klíč, hodnota(x, y)) s novým klíčem, je pro něj vytvořeno pole o velikosti 2. Do tohoto pole se uloží souřadnice bodu pro aktuální klíč.

```
public MapResult(Integer count) {
	key = count;
	x_y = new int [2];
}
```

O vložení souřadnic do pole se stará metoda addValue ().

```
public void addValue(Integer _valuex, Integer _valuey) {
	x_y[0] = _valuex;
	x_y[1] = _valuey;
}
```

Dále obsahuje metody pro získání klíče getKey () a získání jednotlivých souřadnic pro výpočet vzdálenosti getX () a getY ().

5.2 Hadoop implementace

Tato část se zabývá vytvářením aplikací pro Hadoop. Ty se obvykle skládají ze tří tříd:

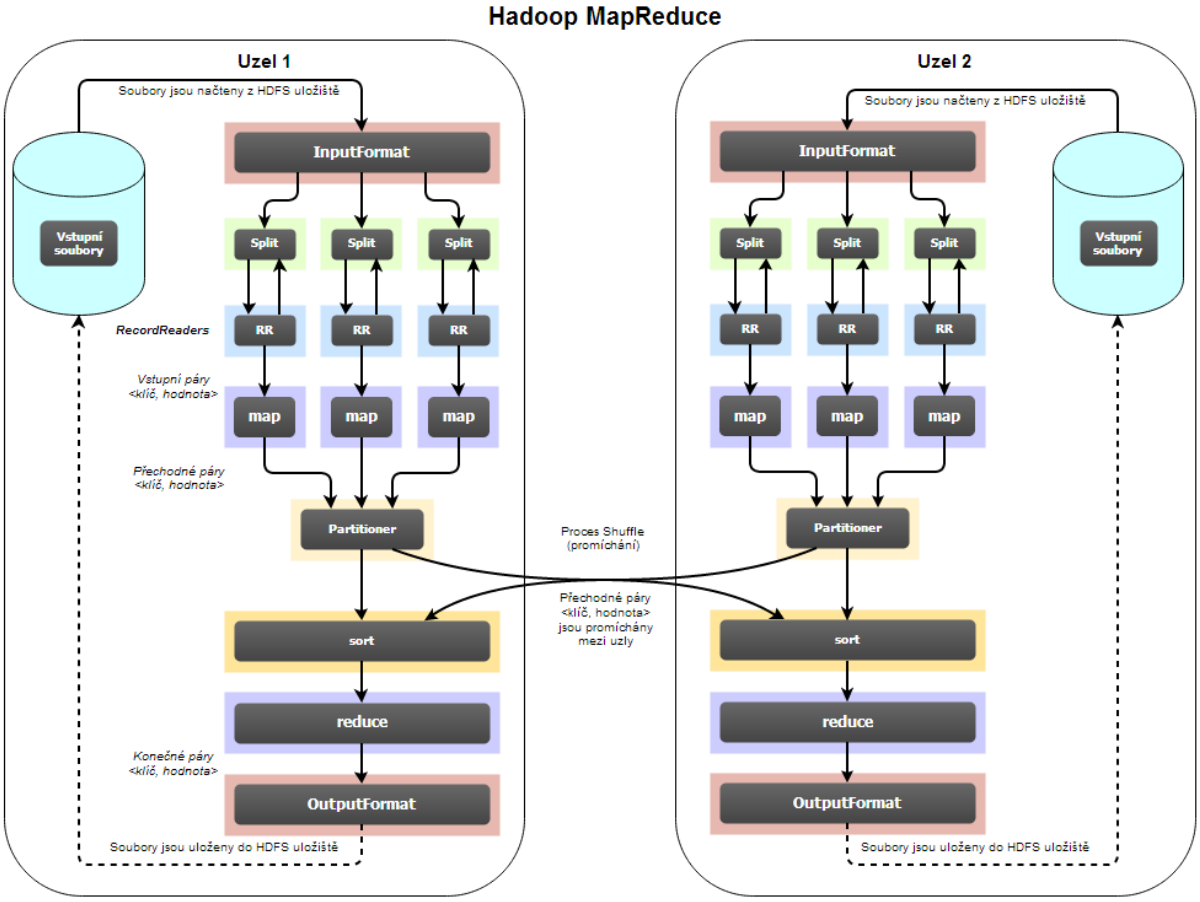
- Mapper – Provádí uživatelem definovanou práci první fáze MapReduce programu. Tato třída rozšiřuje MapReduceBase a implementuje rozhraní Mapper. Toto rozhraní obsahuje 4 generika, která definují typy vstupních a výstupních párů <klíč, hodnota>.
- Reducer – Provádí uživatelem definovanou práci emitovanou z Mapperu. Tato třída také rozšiřuje MapReduceBase a implementuje rozhraní Reducer. A stejně jako Mapper obsahuje 4 generika, která definují typy vstupních a výstupních párů <klíč, hodnota>. První dva parametry definují typ přechodných párů a druhé dva parametry definují konečné typy párů.
- Driver
    - Inicializuje práci a instruuje Hadoop spouštět uživatelský kód.
    - Informace o konfiguraci úlohy je zachycena v objektu *JobConf*.
    - Map a Reduce funkce jsou identifkovány metodami *setMapperClass ()* a *setReducerClass ()*.
    - Datové typy <klíč, hodnota> emitované metodou Reduce jsou označeny jako *setOutputKeyClass ()* a *setOutputValueClass ()*. Ve výchozím nastavení platí tyto datové typy i pro fázi Map. Metody *setMapOutputKeyClass ()* a *setMapOutputValueClass ()* toto výchozí nastavení přepíšou.
    - Zajištění cesty k vstupním a výstupním souborům se provádí pomocí metod *setInputPaths ()* a *setOutputPath ()*.
    - Volání *JobClient.runJob ()* předloží práci MapReduce pro zpracování.

Na rozdíl od vlastní implementace, kde se musela implementovat veškerá funkcionalita, se nyní stačí zaměřit na Map a Reduce metody. V tabulce č. 5. je toto rozdělení práce vypsáno.

Část práce	Vykonává
Konfigurace práce	Uživatel
Rozdělení vstupu a distribuce	Rámec Hadoop
Spuštění jednotlivých Map úloh	Rámec Hadoop
Funkce Map	Uživatel
Míchání	Rámec Hadoop
Seřazení	Rámec Hadoop
Spuštění jednotlivých Reduce úloh	Rámec Hadoop
Funkce Reduce	Uživatel
Kolekce výstupu a uložení do adresáře	Rámec Hadoop

Tab. 5. Rozdělení práce v Hadoop [17]

Na následujícím obrázku č. 24 je zobrazeno blokové schéma zpracování dat v Hadoop MapReduce. Obsahuje pouze dva pracovní uzly, i když v reálném nasazení jich jsou tisíce.



Obr. 24. Blokové schéma zpracování dat v Hadoop

Blokové schéma se skládá z následujících částí:

InputFormat – Definuje, jak jsou rozděleny a čteny vstupní soubory. Tyto soubory jsou následně předány do jednoho nebo více InputSplits. Hadoop poskytuje několik typů vstupních formátů. Voláním metody setInputFormat () se určí, který má být použit. Výchozí formát je *TextInputFormat*, který zpracovává každý řádek v každém souboru jako samostatný záznam. Je vhodný pro neformátovaná data jako log soubory. Další vstupní formát je *KeyValueInputFormat*, který také zpracovává každý řádek jako samostatný záznam a zároveň sám rozdělí řádek na pár <klíč, hodnota>. Poslední *SequenceFileInputFormat* čte speciální binární soubory, které jsou specifické pro Hadoop. Tyto soubory slouží například k dalšímu rychlému načtení dat pro Hadoop mappery.

Splits – Definuje jednotku práce, která obsahuje jeden map úkol v MapReduce programu. Map úkoly můžou načíst celý soubor, ale většinou načtou pouze část souboru. Ve výchozím nastavení je tato velikost 64 MB (nastavení se provádí v souboru Hadoop-site.xml pomocí parametru *mapred.min.split.size*), nebo se může změnit přepsáním parametru v objektu JobConf. Zpracováním souborů do bloků se umožní zpracovávat map úkoly efektivněji a paralelně mezi více uzly.

RecordReader – Načítá bloky ze zdroje, definované vstupním rozdělením a převede je na páry <klíč, hodnota>. Je opakovaně volán na vstup, dokud nejsou spotřebovány všechny vstupní bloky. Každé volání RecordReader vede k volání metody map ().

Mapper – Provádí uživatelem definovanou práci první fáze MapReduce. Každá nová instance běží v samostatném Java procesu pro každý map úkol. Jednotlivé Mapper třídy nejsou záměrně opatřeny mechanismem pro komunikaci mezi sebou. Díky tomu je spolehlivost map úkolů řízena spolehlivostí lokálního počítače. Metoda map () kromě klíče a hodnoty přijímá parametry *OutputCollector* (pomocí metody collect () předkládá páry pro Reduce fázi) a *Reporter* (poskytuje informace o aktuálním úkolu).

Partitioner – Proces přechodu map výstupů do reduce fáze je známý jako *shuffling*. Různé podmnožiny přechodných klíčů jsou přiřazeny Reduce uzlům. Těmto podmnožinám se říká oddíly. Třída Partitioner určuje rozdělení oddílu pro Reduce uzly. Ve výchozím nastavení vypočítává hash hodnotu pro klíč a na základě tohoto výsledku přiřadí oddíl.

Sort – Sada přechodných klíčů na každém uzlu je automaticky seřazena před předložením pro Reducer.

Reduce – Tato instance je vytvořena pro každý Reduce úkol. Provádí uživatelem definovanou práci druhé fáze MapReduce. Jako Map fáze přijímá parametry *OutputCollector* a *Reporter*.

OutputFormat – Páry poskytované tímto kolektorem jsou zapsány do výstupních souborů. Způsob, jakým jsou zapsány, se řídí podle tohoto výstupního formátu a pomocí RecordWriter jsou zapsány do souborů. OutputFormat funguje podobně jako InputFormat. Každý Reducer vytvoří vlastní soubor ve společném adresáři (je nastaven metodou *FileOutputFormat.setOutputPath()*). Tyto soubory jsou pojmenovány part-r-xxxxx, kde xxxxx je id oddílu spojeného s reduce úkolem. Výchozí výstupní formát je *TextOutputFormat,* který píše páry <klíč, hodnota> na jednotlivé řádky textového souboru. Formát *SequenceFileOutputFormat*, který rekonstruuje soubor do stejných datových typů a předkládá data další Map funkci stejným způsobem, jakým byla emitována z Reduce funkce. Poslední *NullOutputFormat* negeneruje žádné výstupní soubory a ignoruje všechny páry předávané kolektorem. To se využívá při explicitním vytváření vlastních výstupních souborů v metodě reduce (). [23,24,28]

5.2.1 Výpočet výskytu slov

Tento typ příkladu nabízí Hadoop jako tutoriál a ukázkový příklad pro snadnější pochopení vývoje Hadoop aplikací. Je k nalezení například v dokumentaci Hadoop verze 1.2.1 na stránkách *https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html*.

Proto jsou použity vlastní uživatelské Map a Reduce třídy. Metody map () a reduce () jsou podobné jako ve vlastní implementaci.

Mapper

Kód načte vstupní soubory jako páry <klíč, hodnota>, zpracuje je a poté tyto páry emituje. Map funkce přijímá *LongWritable* jako klíč, *Text* jako hodnotu a *OutputCollector<Text, IntWritable>* je zodpovědný za zapsání přechodných dat generovaných třídou Mapper.

```
public void map(LongWritable key, Text value, OutputCollector <Text,
IntWritable> output) throws IOException {
	// Načtení slov
	String line = value.toString().toLowerCase();
	// Rozdělení na jednotlivá slova
	String [] words = line.split("\\W");
	// Pro každé slovo předává výskyt 1
	for ( String word : words){
		output.collect(new Text(word), new IntWritable(1));
	}
}
```

Reducer

Třída Reducer načte výstup vygenerovaný třídou Mapper a emituje páry <klíč, hodnota>. Reduce funkce přijímá Text jako klíč, Iterator<IntWritable> pro průchod hodnotami daného klíče a OutputCollector<Text, IntWritable>, který je zodpovědný za konečný výsledek.

```
public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output) throws IOException {
		int sum = 0;
		//pro každé slovo sečte počet výskytů
		while (values.hasNext()) {
			sum += values.next().get();
		}
		output.collect(key, new IntWritable(sum));
}
```

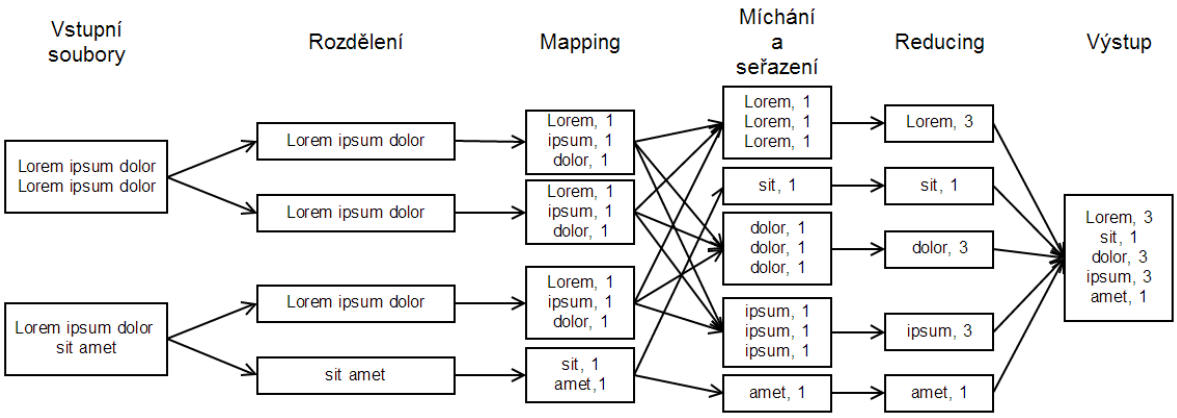
Driver

Kód metody Driver běží na klientském počítači. Je zodpovědný za vytváření konfigurace úlohy a předkládá ji Hadoop clusteru ke zpracování. Obsahuje metodu main (), která přijímá argumenty z příkazového řádku. Nejprve se vytvoří nový objekt JobConf, který předává název Driver třídy. Třída JobConf umožňuje nastavit různé vlastnosti pro Mapper, Reducer, atd.

```
JobConf conf = new JobConf(WordCount.class);
// Nastavení názvu úlohy
conf.setJobName("wordcount");
// Nastavení typu výstupního klíče a hodnoty
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.class);
// Určení Mapper a Reducer třídy
conf.setMapperClass(Map.class);
conf.setReducerClass(Reduce.class);
// Nastavení vstupního formátu
conf.setInputFormat(TextInputFormat.class);
// Nastavení výstupního formátu
conf.setOutputFormat(TextOutputFormat.class);
//nastavení cesty ke konfiguračním souborům
conf.addResource(new Path("/usr/local/hadoop/conf/core-site.xml"));
conf.addResource(new Path("/usr/local/hadoop/conf/hdfs-site.xml"));
// Nastavení vstupní a výstupní cesty
FileInputFormat.setInputPaths(conf, new Path(args[0]));
FileOutputFormat.setOutputPath(conf, new Path(args[1]));
//spuštění úlohy na clusteru
JobClient.runJob(conf);
```

Umístění vstupních a výstupních souborů je v HDFS. Nejprve se do argumentů zadá složka s textovými soubory v HDFS, které mají být zpracovány. Následuje mezera a za ní název složky, do které budou vytvořeny výstupní soubory. Výstupní lokace nesmí před spuštěním programu existovat, protože je vytvořena po provedení programu. Pokud již existuje, pak je program ukončen chybou.

Průběh zpracování WordCount je na obrázku č. 25.



Obr. 25. Průběh Hadoop implementace WordCount

5.2.2 Matice vzdálenosti

Pro výpočet matice vzdáleností byly opět vytvořeny vlastní Map a Reduce třídy. V tomto případě je jako vstupní formát použit *KeyValueTextInputFormat*. Každý řádek vstupních souborů obsahuje klíč, po kterém následuje tabulátor a za ním je souřadnice jednoho bodu.

Mapper

Funkce Map přijímá klíč i souřadnice jako Text, který se používá i pro zapsání přechodných hodnot. Tato třída pouze načte obsah souborů, který přeposílá třídě Reducer.

```
public void map(Text key, Text value,OutputCollector<Text, Text> output,
Reporter reporter)throws IOException {
	// Načtení vstupního souboru pro Reducer
	output.collect(new Text(key),new Text(value));
}
```

Reducer

Funkce reduce jako klíč přijímá hodnotu, která je na začátku každého řádku vstupního souboru. Pro tento klíč se pomocí Iteratoru projdou všechny hodnoty a ty se vloží do ArrayListu. Z tohoto listu jsou potom získávány jednotlivé souřadnice, ze kterých je vypočítána vzdálenost. Výstupní klíč je typu Text a hodnota je typu DoubleWritable.

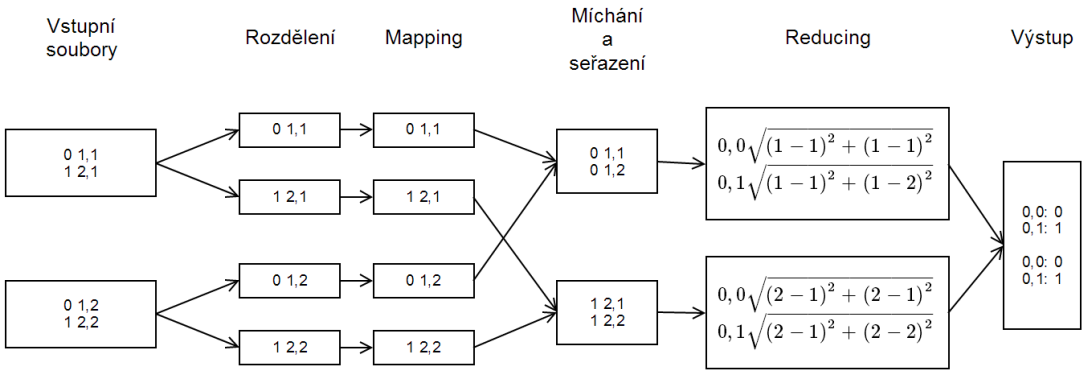
```
// Vytvoření seznamu souřadnic
List<String> coordinates = new ArrayList<String>();
// Procházení hodnot klíče
while(values.hasNext()){
	String coords = values.next().toString();
	// Vložení souřadnic do seznamu
	coordinates.add(coords);
}

for (int i = 0; i < coordinates.size(); i++) {
	x = coordinates.get(i);
	// Rozdělení první souřadnice
	String [] c = x.split(",");
	x1= Integer.parseInt(c[0]);
	y1= Integer.parseInt(c[1]);

	for (int j = 0; j < coordinates.size(); j++) {
		y = coordinates.get(j);
		// Rozdělení druhé souřadnice
		String [] c2 = y.split(",");
		x2= Integer.parseInt(c2[0]);
		y2= Integer.parseInt(c2[1]);
		// Výpočet vzdálenosti
		distance=Math.pow(Math.pow(x1-x2,2)+Math.pow(y1-y2,2),0.5);
		output.collect(	new Text(String.valueOf(i+" "+j+" :")),
				new DoubleWritable(distance));
	}
}
```

Driver

Jelikož výstupní soubor s výslednými hodnotami může dosahovat až několik GB, je jako výstupní formát použit *NullOutputFormat*. Tento formát ignoruje všechny páry předávané kolektorem a nevytváří výstupní soubor. Pokud chceme výstup vytvořit, tak se použije *TextOutputFormat*. Typ výstupu pro klíč je Text a pro hodnoty DoubleWritable, což jsou jednotlivé vzdálenosti.

```
// Nastavení typu výstupního klíče a hodnoty pro Mapper
conf.setMapOutputKeyClass(Text.class);
conf.setMapOutputValueClass(Text.class);
// Nastavení typu výstupního klíče a hodnoty
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(DoubleWritable.class);
// Nastavení vstupního formátu
conf.setInputFormat(KeyValueTextInputFormat.class);
// Nastavení výstupního formátu na Null
conf.setOutputFormat(NullOutputFormat.class);
// Nastavení cesty ke konfiguračním souborům
conf.addResource(new Path("/usr/local/hadoop/conf/core-site.xml"));
conf.addResource(new Path("/usr/local/hadoop/conf/hdfs-site.xml"));
// Určení Map a Reduce třídy
conf.setMapperClass(Map.class);
conf.setReducerClass(Reduce.class);
// Nastavení vstupní a výstupní cesty
FileInputFormat.setInputPaths(conf, new Path(args[0]));
FileOutputFormat.setOutputPath(conf, new Path(args[1]));
```

Průběh zpracování matice vzdáleností je na obrázku č. 26.



Obr. 26. Průběh Hadoop implementace matice vzdáleností

6 TESTOVÁNÍ EFEKTIVITY ŘEŠENÍ

Pro každou implementaci bylo provedeno několik testů, které porovnávají efektivitu vlastní implementace s implementací Hadoop. Testování probíhá jak pro různé dimenze dat, tak i pro různé počty Map a Reduce pracovníků ve vlastní implementace.

Každý test je proveden desetkrát a z jednotlivých časů je vypočítán průměr. V práci jsou uvedeny pouze tyto průměry. Dílčí výsledky testů jsou v příloze na CD v souboru *mapreduce_testy.xlsx*.

6.1 Testovací sestavy

Testování Hadoop aplikací je prováděno v Hadoop clusteru na dvou noteboocích s OS Ubuntu a vlastní aplikace je testována na stroji č. 2 s OS Windows. Obě sestavy jsou popsány v tabulce č. 6.

Notebook	1	2
Hadoop cluster	Master	Slave
Procesor	Intel Core 2 Duo T7300 2 GHz	Intel Core 2 Duo T5870 2 GHz
L2 Cache	4 MB	2 MB
Operační paměť	2 GB	4 GB
Operační systém	Windows 8.1 Pro 32 bitový / Ubuntu 13.10	Windows 8.1 Pro 64 bitový / Ubuntu 13.10

Tab. 6. Testovací sestavy

Jestliže se vlastní implementace testuje s velkými vstupními soubory na počítačích s malou operační pamětí, pak může nastat *Java heap space out of memory* chyba.

6.1.1 Out of Memory chyba

Operační systém, při spuštění Java programu, přidělí část své paměti pro JVM. Ta používá tuto paměť pro všechny své potřeby a část této paměti se nazývá Heap paměť. Pokud je například vytvořen nový objekt pomocí operátoru new, je mu přidělena paměť z této Heap paměti. Výchozí nastavení Heap paměti je pro 32 bitový systém od verze Javy 1.6.0_18. následující. Počáteční maximální velikost Heap paměti je polovina fyzické paměti v případě, že je velikost fyzické paměti do 192 MB nebo jedna čtvrtina z fyzické paměti do velikosti 1 GB této paměti. Pokud není třeba, pak se maximální velikost haldy skutečně nepoužívá. Během inicializace je alokováno menší množství paměti, tedy počáteční Heap velikost. Tato velikost je nejméně 8 MB a nebo 1/64 z fyzické paměti až do velikosti fyzické paměti 1 GB. Pro 64 bitový systém je možno alokovat maximální velikost heap paměti až na 2^{64} bitů (16 exabajtů).

Pokud výchozí nastavení Heap paměti nepostačuje je možné tuto paměť manuálně navýšit. Následující tři příkazy pro JVM se vztahují k velikosti Heap paměti:

-Xms – Nastavení počáteční velikosti Heap paměti

-Xmx – Nastavení maximální velikosti Heap paměti

-Xss – Nastavení velikosti zásobníku vláken

Proměnné uložené v zásobníku jsou viditelné pouze pro dané vlákno, zatímco objekty vytvořené v Heap paměti jsou viditelné pro všechna vlákna. [21,22]

Například příkaz pro nastavení maximální velikosti Heap paměti na hodnotu 1024 MB vypadá následovně: *-Xmx1024m*.

Vstupních data jsou volena tak, aby byla co největší a přitom nenastávala tato chyba při testování.

6.2 Testování výpočtu výskytu slov

K otestování této funkce byl počet textových souborů zvolen tak, aby jejich celková velikost byla 100 MB. Velikosti souborů byly 1, 2, 10, 50 a 100 MB. Při velikosti jednoho souboru například 2 MB, bylo zapotřebí 50 takových souborů. Počet Map pracovníků byl určen počten vstupních souborů a počet Reduce pracovníků byl 1, 2 a 5.

Na obrázku č. 27 jsou 4 tabulky s výsledky WordCount testů.

WordCount

Vlastní implementace

Průměrný čas Map funkce [s]	Map pracovníků					Průměr Reduce
Reduce pracovníků	100	50	10	2	1	
1	133,39	127,20	107,47	142,54	137,38	129,59
2	127,05	141,11	114,56	145,08	164,22	138,40
5	121,23	143,11	109,48	138,59	162,35	134,95
Průměr Map	127,22	137,14	110,50	142,07	154,65	

Průměrný čas Reduce funkce [s]	Map pracovníků					Průměr Reduce
Reduce pracovníků	100	50	10	2	1	
1	0,80	1,45	1,02	1,06	0,90	1,04
2	0,58	0,59	0,71	0,69	0,64	0,64
5	1,62	1,26	1,15	1,05	1,46	1,31
Průměr Map	1,00	1,10	0,96	0,93	1,00	

Celkový průměrný čas [s]	Map pracovníků					Průměr Reduce
Reduce pracovníků	100	50	10	2	1	
1	134,10	128,65	108,50	143,63	137,98	130,57
2	127,78	140,21	115,28	145,79	164,46	138,70
5	122,16	144,38	110,64	139,66	163,96	136,16
Průměr Map	128,01	137,74	111,47	143,03	155,46	

WordCount

Hadoop

Velikost souboru [MB]	1	2	10	5	100
Počet souborů	100	50	10	2	1
Celkový průměrný čas [s]	77,4	69,3	58,7	62,9	64,8

Obr. 27. Výsledky WordCount testů

První dvě tabulky obsahují časy Map a Reduce funkce. V těchto tabulkách jsou vypsány i celkové průměry pro všechny počty Map a Reduce pracovníků. Nejlepší čas je vyznačen zeleně. Ve třetí tabulce jsou vypsány celkové časy jednotlivých měření. Tyto tři tabulky se vztahují k vlastní implementaci. V poslední tabulce jsou časy měření z Hadoop clusteru, které byly prováděny na ta samá data jako ve vlastní implementaci. U celkových časů je zeleně vyznačený nejlepší čas a červeně nejhorší čas.

6.3 Testování výpočtu matice vzdáleností

Testování probíhalo na základě různých počtů vstupních souřadnic bodů. Byly provedeny testy pro 1000, 10000 a 15000 souřadnic. To znamená $1x10^6$, $1x10^8$ a $2{,}25x10^8$ výsledků. K vytvoření vstupních souborů se můžou použít soubory *Generator.java* pro vlastní implementaci a *Generator_format.java* pro implementaci Hadoop.

Na obrázku č. 28 je tabulka s výsledky testů výpočtu matice vzdáleností.

Distance Matrix

Vlastní implementace	Map 1								
	Souřadnic		Výsledků	Souřadnic		Výsledků	Souřadnic		Výsledků
	100x10	1000	1000000	100x100	10000	100000000	100x150	15000	225000000
Reduce	Map funkce [s]	Reduce funkce [s]	Celkový čas [s]	Map funkce [s]	Reduce funkce [s]	Celkový čas [s]	Map funkce [s]	Reduce funkce [s]	Celkový čas [s]
1	0,06	0,65	0,72	0,20	56,18	56,39	0,28	125,02	125,31
2	0,06	0,46	0,53	0,20	32,60	32,81	0,27	71,69	71,97
5	0,07	0,48	0,56	0,22	31,63	31,85	0,28	69,97	70,26
Hadoop	Výsledků	1x 1000^2	3,6	Výsledků	1x 10000^2	122,4	Výsledků	1x 15000^2	272,6
		10x 317^2	4,7		10x 3170^2	121,3		10x 4745^2	271,2

Obr. 28. Výsledky Distance Matrix testů

Jako v předchozím případě jsou zaznamenány celkové časy obou implementací a časy Map a Reduce funkce vlastní implementace. Hadoop implementace obsahuje dva typy testů. Rozdíl je v počtu vstupních souborů. Počet výsledků zůstává téměř stejný. Z výsledných časů vlastní implementace lze zjistit, že většinu práce vykonává funkce Reduce. Měření je proto prováděno pouze s jedním Map pracovníkem. Při rozdělení souřadnic do více souborů vznikne více Map pracovníku, ale jak je vidět na obrázku č. 29, toto řešení nepřináší žádné zlepšení celkových časů. Proto měření s více Map pracovníky nebylo prováděno. Počet Reduce pracovníků byl opět 1, 2 a 5.

Distance Matrix | Vlastní implementace

Reduce 1					
Map 1			Map 2		
Souřadnic		Výsledků	Souřadnic		Výsledků
100x100	10000	100000000	100x50 100x50	5000 5000	100000000
Map funkce [s]	Reduce funkce [s]	Celkový čas [s]	Map funkce [s]	Reduce funkce [s]	Celkový čas [s]
0,20	56,18	56,39	0,21	55,72	55,94

Obr. 29. Rozdíl časů při změně počtu Map pracovníků

ZÁVĚR

Cílem diplomové práce bylo prostudovat a naimplementovat metodu MapReduce a aplikovat ji na zvolený problém. Tuto metodu jsem si vybral, jelikož velká data celosvětově představují rychle rostoucí mnohamiliardový trh, který nabízí atraktivní pracovní příležitosti pro odborníky se správnými znalostmi v oblasti technologií, analýzy dat, komunikace a konkrétních průmyslových odvětví. Pro porovnání efektivity vlastní implementace metody MapReduce byl vybrán nejrozšířenější open source projekt Hadoop. Byly popsány jeho jednotlivé části a princip fungování. Součástí práce bylo i porovnat projekt Hadoop s jeho dalšími distribucemi. Mezi významné distributory patří například Cloudera, DataStax, MapR nebo i společnosti jako IBM, Intel a Microsoft. Jednotlivé distribuce se od sebe liší například verzí Hadoop, kterou obsahují, nebo souvisejícími projekty, které poskytují další nástroje pro práci s daty. Tato práce slouží i jako ucelený návod pro nasazení a spuštění Hadoop. Návod obsahuje požadavky před samotnou instalací a popis instalace s jedním i více uzly. Je vysvětleno, jak se spustí Hadoop cluster, jak se dostat do Hadoop webového rozhraní, jaké jsou příkazy pro interakci s HDFS nebo jak vytvořit a nastavit Eclipse plugin, který poskytuje nástroje pro snadnou práci s MapReduce.

Implementace metody MapReduce se vztahovala na dva úkoly. Výpočet matice vzdáleností pro velký počet bodů a počítání výskytu slov v textových souborech. Pro vlastní implementaci metody MapReduce byl použit jazyk Java a vývojové prostředí Eclipse. Tato implementace pomocí vláken simuluje funkcionalitu výpočetního prvku MapReduce projektu Hadoop. V kapitole číslo 5 je popsán postup, jak jsou jednotlivé data zpracována, rozdělena a ukládána i s důležitými ukázkami kódu, které se o tuto práci starají. U každé implementace je obrázek, znázorňující průběh zpracování dat. Hadoop implementace zpracovává každý řádek jako samostatný proces, zatímco ve vlastní implementaci jsou zpracovávány celé soubory. Také se liší předání přechodných párů do Reduce fáze, kdy se ve vlastní implementaci neprovádí rozdělení bloků pomocí vypočítání hash funkce.

Pro každou implementaci byly provedeny testy, které porovnávají efektivitu vlastní implementace s implementací na rámci Hadoop. Pro snadnější testování vlastní implementace bylo vytvořeno grafické rozhraní MapReduce.jar. Výpočet výskytu slov byl u obou implementací testován na textových souborech o celkové velikosti 100 MB. Z výsledných tabulek lze zjistit, že většinu práce ve vlastní implementaci provádí funkce Map, která se stará o načtení veškerého obsahu souborů a pro každé slovo vytvoří seznam s jeho výskyty. Nejlepší časy tato funkce měla s 10 vstupními soubory (10 Map pracovníků) o velikosti 10 MB a to v průměru za 110,5 sekund. Reduce funkce pouze pro každé slovo sečte počet jeho výskytů. Tato operace se nejrychleji provedla při rozdělení přechodných dat pro 2 Reduce pracovníky a to v průměru za 0,64 sekund. Celkový čas byl tedy dán hlavně Map funkcí a nejrychleji proběhl v průměru za 108,5 sekund při 10 Map pracovnících a 1 Reduce pracovníkovi. Nejhorší čas 163,9 sekund byl naměřen u jednoho 100 MB souboru, kdy se zpracování souboru

nerozdělilo mezi více pracovníků. Hadoop implementace WordCount si nejlépe poradila s 10 soubory v průměru za 58,7 sekund a nejhůře si vedla při stech 1 MB souborech a to v průměru za 77,4 s. Hadoop implementace v tomto případě předčí vlastní implementaci. Tyto rozdíly jsou dány díky již zmiňovanému rozdílnému způsobu načítání a zpracování vstupních dat.

Testování výpočtu matice vzdáleností probíhalo pro 1000, 10000 a 15000 souřadnic. V tomto případě většinu práce obstarává funkce Reduce, ve které se provádí výpočty jednotlivých vzdáleností. Map funkce pouze načte souřadnice ze souboru. Proto jsou měření prováděna pouze s jedním Map pracovníkem. U všech měření je vidět zlepšení časů téměř o polovinu při přechodu z jednoho na dva Reduce pracovníky. Další přechod na pět pracovníků již takové zlepšení nepřináší, to je zapříčiněno především počtem jader procesoru testovací sestavy. Pro $1x10^6$ vzdáleností byl nejlepší celkový čas 0,53 sekund při 2 Reduce pracovnících a z toho 0,48 sekund trvala Reduce fáze. Výpočet $1x10^8$ vzdáleností byl nejrychleji za 31,6 sekund a $2,25x10^8$ vzdáleností za 70,3 a to u obou při 5 Reduce pracovnících. Hadoop implementace měla průměrné celkové časy 3,6 sekund, 122,4 sekund a 272,6 sekund pro souřadnice bodů se stejným klíčem v jednom souboru. Výsledné časy pro Hadoop implementaci byli tentokrát větší. Proto byl proveden další test, ve kterém byly souřadnice rozděleny do 10 souborů s 10 různými klíči. Toto rozdělení však nepřineslo žádné zlepšení. Rozdíl oproti vlastní implementaci je například ve formátu souřadnic ve vstupních souborech. Hadoop implementace zpracovává každý řádek jako samostatný záznam a zároveň rozděluje řádky na páry (klíč, hodnota). Vkládání hodnot nejprve do ArrayListu v Reduce fázi a jeho následné procházení také není příliš účinné řešení.

Vlastní implementace slouží především pro správné porozumění fungování a principu metody MapReduce a její výkon je dán výkonem stroje na kterém běží. Není vytvořena pro zpracování velkých dat. Naproti tomu od momentu, kdy se velikost dat pohybuje ve stovkách gigabajtů až v terabajtech, začíná být zajímavé použití Hadoop. Například při analýze sociálních sítí, které jsou specifické svojí nestrukturovaností a dynamikou. Nebo při vědeckých experimentech, které generují obrovské množství dat náročných na výpočty. Hadoop však není vhodnou platformou pro všechny úkoly. Například při transakčních úlohách, kde nemůže konkurovat relačním databázím. Zřejmě největším omezením je v současnosti nedostatek zkušených odborníků, kteří by měli být schopni analyzovat velké datové objemy, používat statistické a matematické metody, a také výsledky vizualizovat. Zároveň však musí být schopni porozumět business problematice a vědět, co konkrétně v datech hledat. Jedná se o roli na pomezí statistika, analytika a vývojáře.

Tato práce poskytuje všeobecné informace o programovacím modelu MapReduce, o projektu Hadoop a jeho nasazení. Slouží jako návod pro vytváření Hadoop aplikací i jako návod pro implementaci vlastní metody MapReduce.

SEZNAM POUŽITÉ LITERATURY

[1] Big data a jejich zpracování [online]. [2013-09-25]. Dostupný z www: <http://www.root.cz/clanky/big-data-a-jejich-zpracovani/>.

[2] Big data Nové způsoby zpracování a analýzy velkých objemů dat [online]. [2013-09-25]. Dostupný z www: <http://www.systemonline.cz/clanky/big-data.htm>.

[3] Big data? Pro české podniky zatím spíš velká neznámá [online]. [2013-09-25]. Do stupný z www:<http://www.systemonline.cz/zpravy/big-data-pro-ceske-podniky-zatim-spis-velka-neznama-z.htm>.

[4] Velká data – nové příležitosti a výzvy neznámá [online]. [2013-09-25]. Dostupný z www: <http://www.systemonline.cz/clanky/velka-data-nove-prilezitosti-a-vyzvy.htm>.

[5] Welcome to Apache™ Hadoop®! [online]. [2013-09-25]. Dostupný z www: <http://hadoop.apache.org>.

[6] Disco massive data - minimal code [online]. [2013-09-25]. Dostupný z www: <http://discoproject.org>

[7] Meguro, a simple Javascript Map/Reduce Framework [online]. [2013-09-25]. Dostupný z www: <http://www.sevenforge.com/meguro/>.

[8] Octopy Easy MapReduce for Python [online]. [2013-09-25]. Dostupný z www: <https://code.google.com/p/octopy/>.

[9] Mars: A MapReduce Framework on Graphics Processors [online]. [2013-09-25]. Dostupný z www: <http://www.cse.ust.hk/gpuqp/Mars.html>.

[10] Running Hadoop on Ubuntu Linux (Single-Node Cluster) [online]. [2014-01-21]. Dostupný z www: <http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/>.

[11] Running Hadoop on Ubuntu Linux (Multi-Node Cluster) [online]. [2014-01-21]. Dostupný z www: <http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/>.

[12] Single Node Setup [online]. [2014-01-21]. Dostupný z www: <http://hadoop.apache.org/docs/r1.2.1/single_node_setup.html>.

[13] Jeffrey Dean and Sanjay Ghemawat; *MapReduce: Simplified Data Processing on Large Clusters*, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.

[14] Alex Holmex; *Hadoop in Practice*, Manning Publications Co.; 2012. ISBN: 978-1-61729-023-7.

[15] Chuck Lam; *Hadoop in Action*, Manning Publications Co.; 2011. ISBN: 978-1-93518-219-1.

[16] Eric Sammer; *Hadoop Operations*, O'Reilly Media, First Edition, 2012, ISBN: 978-1-449-32705-7.

[17] Jason Venner; *Pro Hadoop*, Apress, 2009, ISBN-13: 978-1-4302-1942-2.

[18] Jimmy Lin and Chris Dyer; *Data-Intensive Text Processing with MapReduce*, University of Maryland, College Park Manuscript prepared April 11, 2010.

[19] Tom White; *Hadoop: The Definitive Guide*, O'Reilly Media; Third Edition, 2012, ISBN-13: 978-1449311520.

[20] James E. Gentle; *Matrix Algebra: Theory, Computations, and Applications in Statistics*. Springer-Verlag, 2007. ISBN 0-387-70872-3.

[21] Update Release Notes [online]. [2014-01-21]. Dostupný z www: <http://www.oracle.com/technetwork/java/javase/6u18-142093.html>.

[22] 10 points about Java Heap Space or Java Heap Memory [online]. [2014-01-21]. Dostupný z www: <http://javarevisited.blogspot.cz/2011/05/java-heap-space-memory-size-jvm.html>.

[23] Yahoo! Hadoop Tutorial [online]. [2014-01-21]. Dostupný z www: <https://developer.yahoo.com/hadoop/tutorial/index.html>.

[24] MapReduce Tutorial [online]. [2014-01-21]. Dostupný z www: <https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html>.

[25] Hadoop Eclipse Plug-in [online]. [2014-01-21]. Dostupný z www: <http://wiki.apache.org/hadoop/EclipsePlugIn>.

[26] Když se řekne „Hadoop“ [online]. [2014-01-21]. Dostupný z www: <http://www.linuxexpres.cz/software/kdyz-se-rekne-hadoop>.

[27] Srinath Perera, Thilina Gunarathne; *Hadoop MapReduce Cookbook*, Packt Publishing, First Edition, 2013, ISBN: 978-1-84951-728-7.

[28] Donald Miner and Adam Shook; *MapReduce Design Patterns*, O'Reilly Media, First Edition, 2013, ISBN: 978-1-449-32717-0.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: PŘEHLED DISTRIBUCÍ

	Distributor	Apache	Cloudera	DataStax	EMC	Hortonworks	IBM	Intel	MapR	Microsoft
	Název distribuce	Hadoop	CDH4	Enterprise	Greenplum HD	Data Platform	InfoSphere BigInsights	Distr. For Hadoop	M5 Edition	HDInsight Server
	Verze distribuce	2.4.1	4.7.0	4.5.1	1.2.2.3	2.1	3.0	2.3	4.0.1	3.1
	Verze Hadoop	2.4.1	2.0.0	1.0.4.13	2.x	2.4.0	2.1.2	1.0.3	2.4.1	2.4.0
Integrované subprojekty	Cassandra	x		x						
	Chukwa	x								
	Flume	x	x			x	x	x	x	
	Hbase	x	x		x	x	x	x	x	
	HDFS	x	x	x	x	x	x	x	x	
	Hcatalog	x	x			x	x		x	x
	Hive	x	x	x	x	x	x	x	x	x
	Jaql						x			
	Mahout	x	x	x	x	x		x	x	x
	MapReduce	x	x	x	x	x	x	x	x	
	Oozie	x	x			x	x	x	x	x
	Pig	x	x	x	x	x	x	x	x	x
	Sqoop	x	x	x		x	x	x	x	x
	Zookeeper	x	x		x	x	x	x	x	x
Správa	Clusterů	Apache Ambari	Cloudera Manager	DataStax OpsCenter	Greenplum HD Command Center	Apache Ambari	BigInsights Console	Intel Manager	MapR Control Systém	HDInsight Dashboard/ Ambari
	RT/LL data management		Impala				x	x		
Integrace	POSIX přístup						x		x	
	SQL přístup		Impala		v Pivotal HD	Teradata SQL-H	x			PolyBase
Nasazení	Vysoká dostupnost	2.x	x			x	x		x	
	Certifikace pro OS	Cross platform	RHEL, CentOS, Oracle L., SLES, Ubuntu, Debin	CentOS, Debian, Mac, Oracle, Red Hat, SUSE, Ubuntu	RHEL, CentOS	RHEL, CentOS, SLES, WIN	RHEL, SLES	RHEL, CentOS, Oracle L., SLES	RHEL, CentOS, SLES, Ubuntu	Windows
	Appliance		+Oracle		x	+Teradata	x	x		x

Obr. 30. Přehled distribucí

Vysvětlivky:

- Integrované sub-projekty – V rámci příslušné distribuce Hadoop.
- Real time / low latency data management – Podpora zpracování dat v reálném čase.
- Vysoká dostupnost – Žádná porouchaná komponenta nesmí způsobit výpadek systému.
- POSIX přístup – Kompatibilní POSIX rozhraní pro HDFS.
- SQL přístup – Nativní SQL dotazovací rozhraní pro Hadoop.
- Appliance – Možnost dodání komplexního optimalizovaného řešení včetně hardwarových komponent.

PŘÍLOHA P II: OBSAH SOUBORU .BASHRC

```
# Set Hadoop-related environment variables
export HADOOP_HOME=/usr/local/hadoop
# Set JAVA_HOME (we will also configure JAVA_HOME directly for Hadoop later on)
export JAVA_HOME=/usr/lib/jvm/java-6-sun
# Some convenient aliases and functions for running Hadoop-related commands
unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -ls"
lzohead () {
    hadoop fs -cat $1 | lzop -dc | head -1000 | less
}
# Add Hadoop bin/ directory to PATH
export PATH=$PATH:$HADOOP_HOME/bin
```

PŘÍLOHA P III: UKÁZKA DFSADMIN -REPORT

```
hduser@martin-HP-Compaq-6710b:/usr/local/hadoop$ hadoop dfsadmin -report
Warning: $HADOOP_HOME is deprecated.

Configured Capacity: 63136894976 (58.8 GB)
Present Capacity: 48109428736 (44.81 GB)
DFS Remaining: 48105246720 (44.8 GB)
DFS Used: 4182016 (3.99 MB)
DFS Used%: 0.01%
Under replicated blocks: 2
Blocks with corrupt replicas: 0
Missing blocks: 0

-------------------------------------------------
Datanodes available: 2 (2 total, 0 dead)

Name: 192.168.1.100:50010
Decommission Status : Normal
Configured Capacity: 37255593984 (34.7 GB)
DFS Used: 2088960 (1.99 MB)
Non DFS Used: 8855142400 (8.25 GB)
DFS Remaining: 28398362624(26.45 GB)
DFS Used%: 0.01%
DFS Remaining%: 76.23%
Last contact: Mon Jan 27 15:27:20 CET 2014


Name: 192.168.1.116:50010
Decommission Status : Normal
Configured Capacity: 25881300992 (24.1 GB)
DFS Used: 2093056 (2 MB)
Non DFS Used: 6172323840 (5.75 GB)
DFS Remaining: 19706884096(18.35 GB)
DFS Used%: 0.01%
DFS Remaining%: 76.14%
Last contact: Mon Jan 27 15:27:20 CET 2014
```

Obr. 31. Příkaz dfsadmin –report

PŘÍLOHA P IV: PŘÍLOHY NA CD/DVD

- Složka MapReduce
  - Obsahuje zdrojové kódy vlastní implementace.
- Složka Generator
  - Obsahuje zdrojové kódy k vygenerování vstupních souborů pro úkol výpočtu matice vzdáleností.
- Složka Hadoop
  - Obsahuje zdrojové kódy Hadoop implementací.
- Soubor mapreduce_testy.xlsx:
  - Obsahuje všechny tabulky a výsledné časy testů.
- Soubor MapReduce.jar
  - Aplikace vlastní implementace MapReduce.
- Soubor Diplomová_práce.pdf:
  - Kompletní diplomová práce ve formátu pdf.
- Soubor Přílohy.pdf
  - Přílohy diplomové práce.
- Soubor hadoop-eclipse-plugin-1.2.1.jar
  - Jedná se o vytvořený Hadoop plugin pro vývojové prostředí Eclispe.